

STPAcc: Structural TI-based Pruning for Accelerating Distance-related Algorithms on CPU-FPGA Platforms

Yuke Wang, Boyuan Feng, Gushu Li, Lei Deng, *Member, IEEE*, Yuan Xie, *Fellow, IEEE*, and Yufei Ding
University of California, Santa Barbara, USA

Abstract—As a promising solution to boost the performance of distance-related algorithms (e.g., K-means and KNN), FPGA-based acceleration attracts lots of attention, but also comes with numerous challenges. In this work, we propose, STPAcc, an optimization framework based on structural triangle-inequality (TI) based pruning (STP) for accelerating distance-related algorithms on CPU-FPGA platforms. STPAcc provides a domain-specific language to unify distance-related algorithms effectively, a structural TI-based pruning strategy to remove unnecessary distance computations, a coarse-grained workload partitioning and mapping strategy to fully exploit the potentials of the CPU-FPGA platform, and fine-grained hardware optimizations to further improve performance on the FPGA. Intensive experiments show that STPAcc designs achieve $31.42\times$ speedup and $99.63\times$ better energy efficiency on average over standard CPU-based implementations.

Keywords: Machine Learning Algorithms; OpenCL Programming; FPGA Acceleration.

I. INTRODUCTION

Distance calculations are essential to many algorithms across various domains: data analytics [37], [51], [4], graph analysis [24], [14], digital imaging [15], [36], and scientific simulations [55], [45]. For example, the commonly used clustering algorithm, K-means [42], is an iterative algorithm that computes distances between every data point and each of K cluster centers to find the closest cluster center of each point. K-Nearest Neighbor (KNN) [25] finds the K points in a target set that are closest to every point in the query set. N-body simulation [58] computes the distances between every particle and all its neighbors in every time step, to simulate the interplay of particles and their resulting movements. Other examples include point-to-point shortest path [17] in graph analysis, 3D image construction [10] for digital imaging, etc.

These distance-based algorithms are indispensable for analyzing data in many scientific and engineering disciplines. Clustering and N-body simulation, for instance, are announced as two of “the seven giants of statistical data analysis” by the National Academy of Sciences [12]. K-means and KNN are rated among the Top-10 most influential data mining

algorithms [63]. The need to improve the computing efficiency of these algorithms has never been satisfied. It is evidenced by a large number of research papers published in the premium venues in those domains. For instance, in the recent 10 years of top machine learning or data mining conferences, there are more than 20 papers [20], [23], [27], [46], [59], [54] on pure algorithmic optimization for accelerating only K-means.

However, pure algorithmic optimization is clearly not enough. Co-optimizations from both algorithmic and hardware side are expected to offer faster and more energy-efficient solutions. In particular, FPGA-based platforms gain lots of interest from both industry and research field recently. Compared to standard CPU and GPU platforms, which often suffer from either inferior performance or high-power consumption, FPGA-based implementations could potentially offer better Pareto-optimal solutions, especially in terms of energy efficiency, making them promising solutions for emerging computing platforms with limited power budgets. This being said, accelerating these distance-related algorithms on FPGAs requires non-trivial efforts, including hardware expertise, time, and monetary cost. While existing research efforts try to ease this process [31], [53], [39], [38], [44], [56], they inevitably fall short in at least one of the following aspects.

Rely on problem-specific designs and optimizations while missing effective generalization. Most of the previous hardware designs and optimizations [31], [39], [53], [38] are heavily coded for a specific algorithm and some specific inputs. There is no such unified abstraction to formalize the definition and optimization of distance algorithms systematically. For example, work from [31] manually optimizes K-means on a specific FPGA board towards very small scale bio-microarray data. This solution can not be adapted to inputs of varying sizes or shared across different distance-related algorithms.

Lack of algorithm-hardware co-design. Previous algorithmic [22], [19] and hardware optimizations [39], [53], [31], [8], [38] are applied separately instead of being combined collaboratively. Existing algorithmic optimizations, most of which are based on a fine-grained triangle-inequality (TI) based filtering strategy [22], [19], [18], [9], could save many distance computations, but also incur high computation irregularity and memory overhead. On the other hand, existing hardware accelerations rather rely on those standard distance-related algorithms (w/o any algorithm-level optimization) to minimize the efforts in the hardware implementations.

Yuke Wang, Boyuan Feng, Yufei Ding are with the Department of Computer Science, University of California, Santa Barbara, USA. (email: yuke_wang@ucsb.edu, boyuan@cs.ucsb.edu, yufeiding@cs.ucsb.edu). Gushu Li, Lei Deng, and Yuan Xie are with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, USA. (email: gushuli@ucsb.edu, leideng@ucsb.edu, and yuanxie@ucsb.edu). Yuke Wang is the corresponding author.

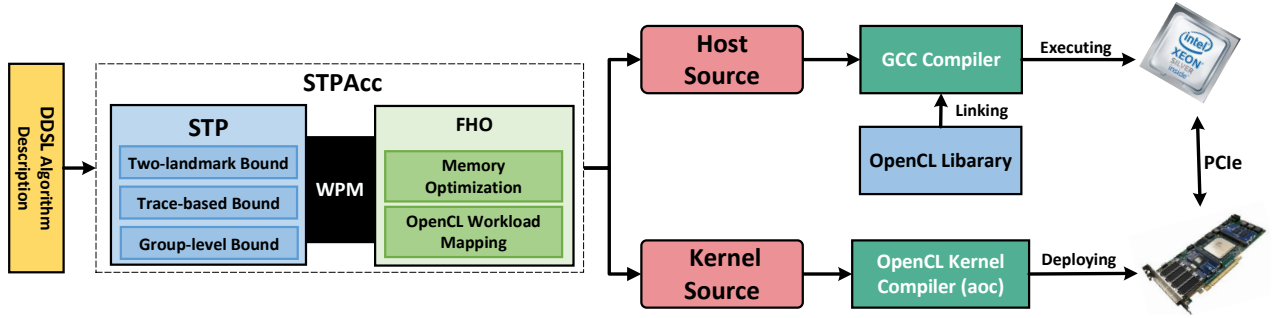


Fig. 1: STPAcc Overview.

Count on FPGAs as the only source of acceleration.

Previous work [56], [44], [29], [35], [31], [38] places the whole algorithm on the FPGA accelerator without considering the assists from the computing resource on the host CPU. They thus miss the full performance benefits from the heterogeneous CPU-FPGA computing paradigm, for example, using the CPU for complex logic and control-intensive operations while only offloading the compute-intensive tasks to the FPGA. As a result, their designs are usually limited by the on-chip memory and computing elements of the FPGA boards.

To this end, we build the first optimization framework, namely STPAcc (Figure 1), that can automatically optimize and generate high-performance and power-efficient designs of distance-related algorithms on the CPU-FPGA computing platform. Specifically, STPAcc takes the distance-related algorithm in our distance-domain specific language as the put, then apply a set of optimizations, including structural TI-based pruning, coarse-grained workload partitioning and mapping, and fine-grained hardware optimization. Later, STPAcc will generate the implementation for running on CPU host and FPGA devices with the corresponding compiler tool chains (e.g., GCC). In short, STPAcc comes with a 1) library-based, domain-specific language, with which end users could focus on the high-level semantics of their applications rather than low-level implementation details; 2) compiler-based automatic transformation prototype, which would automatically generate highly-optimized implementations for different distance-related algorithms towards various sizes of input datasets. The following summarizes key technical contributions.

- STPAcc provides a **distance domain-specific language (DDSL)**, which abstracts the commonalities across different distance-related problems and offers several language constructs (e.g., *Definition*, *Operation*, and *Control* construct) for describing distance-related algorithms. Besides, DDSL follows a concise yet expressive C language style, offering more flexibility in low-level control and further FPGA accelerator optimization.
- STPAcc features a novel **Structural TI-based Pruning (STP)** scheme to facilitate efficient algorithm-hardware co-optimization. STP generalizes the traditional, fine-grained TI-based optimization with *two-landmark*, *trace-based*, and *group-level* bound computations. These together eliminate redundant distance computations, while maintaining the computation regularity for ease of hardware acceleration.

- STPAcc leverages a **coarse-grained workload partitioning and mapping (WPM)** to fully utilize the CPU-FPGA computing paradigm. Our key observation is that STP optimization will lead to tasks with different computing patterns. Thus, we distribute the tasks with complex operations and execution dependency, such as *distance filtering*, to the CPU, while assigning those with simple and vectorizable operations, such as *distance computations*, to the FPGA.
- STPAcc applies a set of **fine-grained hardware optimizations (FHO)** to further accelerate remaining computations on the FPGA, such as kernel-level memory optimizations to reduce inter-group and intra-group memory irregularity, and a parallelized OpenCL kernel architectural design to support STP-featured workloads and improve the overall performance.

Intensive experiments on several popular algorithms across a wide spectrum of datasets show that STPAcc-generated CPU-FPGA designs achieve $31.42\times$ speedup and $99.63\times$ better energy-efficiency on average compared with standard CPU-based implementations.

II. RELATED WORK

A. Structural Pruning

Pruning strategy, as its name suggests, aims to “trim off” some unnecessary computations without compromising the functionality of the applications. Previous research in DNN acceleration has witnessed its success in removing the redundancy computation in NN models [47], [65], [66], [41], [43]. For example, Niu et al. [66] propose a DNN weight pruning to avoid unnecessary weight computation to facilitate the real-time execution of DNNs on resource-limited mobile devices. Liu et al [41] automate the DNN pruning for achieving an ultra-high compression rate. However, fine-grained pruning on DNN models would cause severe computation irregularity problems leading to inferior performance. To leverage the massive parallelism and pipeline on modern general processors and accelerators, a structural pruning strategy [47], [65] has been proposed to maximize the performance gains of pruning benefits on the modern hardware. While sharing some similarities with these studies on DNN, our work presents a unique set of challenges. Specifically, traditional DNN is a model-based learning and its structural pruning mainly focuses on model-level redundancy, which is static and input-agnostic, and can

be determined offline before taking any inputs. Distance-related algorithms, on the other hand, are instance-based learning. The best pruning strategy would depend on the input information (e.g., the data size and distribution), thus, making such pruning more challenging.

B. TI-Optimization

TI-based optimization [22], [19], [18], [9], [62], [50] aims at pruning computation-expensive distance computations with cheaper TI-based bound computations, but still guaranteeing the same final results as the original algorithms. Take K-means as an example, many previous TI-based optimizations such as Elkans' K-means [22] and Yinyang K-means [19], have validated that their optimizations can always guarantee the same clustering results (e.g., both point assignments and final cluster centers) as the standard version (with no distance computation removed). The key behind such a quality guarantee is that distance computations can only be eliminated when we are 100% sure that they will not change the final results. Besides such accuracy guarantees, advantages of TI-based optimizations also include better flexibility and scalability compared to other algorithm-level pruning strategies, such as KD-tree based methods [16], [53]. In particular, the performance advancement of these TI-based work [19], [18], [9] is more robust in general across datasets with a wide range of input sizes and dimensions. As such, we focus on structural TI-based pruning optimization at algorithmic optimization.

C. Hardware Acceleration

As a raising technique, FPGA-based acceleration becomes a popular solution for speeding up data analytics, and numerous FPGA designs [31], [56], [53], [38] have been proposed and studied. For example, work from [31], [53], [39] targets KNN FPGA acceleration, while researchers from [38], [44], [56] focus on K-means. Despite significant performance improvements introduced by these methods, they mostly fail to incorporate algorithmic optimizations in the hardware design: they directly port the standard distance-related algorithms to FPGAs and only apply hardware-level optimization. Even worse, these previous designs [39], [31] usually assume that the dataset can be fully fit into the FPGA on-chip memory, and are only evaluated on a limited number of small datasets. For example, in [39], K-means is evaluated on a micro-array dataset with only 2,905 points. These designs often encounter portability issues when transferring to different settings.

In addition, previous work largely focuses on the traditional hardware design flow, which requires a long implementation cycle and huge manual efforts. For example, work from [29], [31], [8], [53], [57], [35], [28], [30] builds the design based on VHDL/Verilog design flow, which requires hardware expertise and over months of arduous development. These "hard-coded" optimizations also create difficulties for a fair comparison across different designs, which hamper future studies in this direction. In contrast, STPAcc design flow brings significant advantages of programmability and flexibility from its high-level OpenCL programming model, which minimizes the user efforts in the tedious hardware design process.

III. DISTANCE DOMAIN-SPECIFIC LANGUAGE

To accommodate various distance-related algorithms while maintaining sufficient input adaptability, STPAcc defines a Distance Domain-Specific Language (DDSL). The major rationale of using DDSL is to combine the algorithm and hardware optimization in a unified way meanwhile simplifying design optimization and hardware mapping. Previous high-level synthesis (HLS)-based solutions exposing more involved hardware details complicate the algorithm optimizations while demanding lots of user efforts in design and implementation.

By summarizing different distance-related algorithms, we make two key observations. First, distance-related algorithms would differ in their interested distances. For example, the KNN algorithm focuses on the Top-k closest neighbors of each point, while the N-body simulation aims at finding the neighbors within a certain range in 3D space; second, distance related algorithms would differ in their control flow. For example, the K-means clustering operates iteratively, while KNN algorithm only executes once.

Based on these observations, we propose DDSL construct-based design, which consists of three components (*Definition*, *Operation*, and *Control*). In particular, the definition construct (e.g., `dVar`) defines the input data; the operation construct defines different operations on data points and distances (e.g., `STPAcc_Dist_Select` handles algorithms with different interested distances); the control construct (`STPAcc_Iter`), handles algorithms with different execution flows. We detail these constructs in Table I.

A. Examples

In this subsection, we showcase the DDSL effectiveness by introducing three popular distance-related algorithms and their corresponding DDSL representations.

K-means [42], [33], [34], [11], [52] clusters a set of points into several groups in an iterative manner. At each iteration, it first computes the distances between each point and all clusters and assigns each point to its closest cluster. Then it updates the cluster centroids based on the average position of their inside points. The DDSL of K-means is shown as follow:

```
STPAcc_Iter(S){
  S = false;
  STPAcc_Comp_Dist(pSet, cSet, distMat,
    D, "unwegL1", null);
  STPAcc_Dist_Select(distMat, idPtr, idArr,
    TopK, "small");
  STPAcc_Update(cSet, pSet, idPtr, idArr, S);
}
```

where TopK is the Top-K value of interest (here, TopK=1 for K-means); D is the size of dimension; psize is the number of points and csize is the number of clusters; pSet is the point-set; cSet is the cluster-set; disMat is the point-to-cluster distance matrix; idPtr and idArr is the neighbor-IDs of each point; S is the status of iteration.

KNN-join [5], [26], [64] finds the Top-K nearest neighbor points for each point in the source set from the target set. It first computes the distances between each source point and all the target points. Then it ranks the K-smallest distances for each source point and gets its closest Top-K target points. The DDSL of KNN-join is shown as follow:

TABLE I: Constructs Details.

Constructs	Categories	Details
DVar [varName] DType [Initial_Value]; DSet [setName] DType [size] [dim];	Data Construct	It leverages DVar to define a single variable, DSet to define the dataset, and the DType to notate the type of the defined variable or dataset.
STPAcc_Comp_Dist (Input p1, Input p2, Output disMat, Dim dim, Met mtr, Weg wArr)	Distance-compute Construct	It measures the exact distance between two different data points. p1, p2: Input data matrix ($n_1 \times d$, $n_2 \times d$); disMat: Output distance matrix ($n_1 \times n_2$); dim: Dimensionality of data points (d); mtr: Distance metric (weg unweg)(L1 L2); wArr: Weight array for weighted distance ($1 \times d$).
STPAcc_Dist_Select (Input distMat, Output idPtr, Output idArr, Range ran, Scope scp)	Distance-selection Construct	It selects the distances and IDs of neighbors of interest for each point. idPtr: ID pointers (1-D array of pointers for indexing neighbor-IDs of each point in IdArr); idArr: ID array (1-D array with neighbor-IDs for each point); ran: the number of neighbors of interest (e.g., K-means, KNN) or distance threshold (e.g., N-body Simulation); scp: the Top-K smallest or largest values.
STPAcc_Update (Update upVar, Input {p1, ..., pm}, Status S)	Data-update Construct	upVar: Input data/dataset to be updated; p1, ..., pm: Additional information used in update; S: Status of update operation.
STPAcc_Iter (maxIterNum exitCond){ subConstruct sc_1; subConstruct sc_2; ... subConstruct sc_n; }	Control Construct	It is used to describe the distance-related algorithms that require to run multiple iterations. Iteration construct requires users to provide either the maximum number of iterations or other exit conditions.

```

STPAcc_Comp_Dist(pSet, cSet, distMat,
                D, "unwegL1", null);
STPAcc_Dist_Select(distMat, idPtr, idArr, TopK,
                  "small");

```

where TopK is the Top-K value of interest.

N-body Simulation [49], [32] mimics the particle movement within a certain range of 3D space. At each time step, distances between each particle and its neighbors (within a radius R) are first computed, and then the acceleration and the new position of each particle will be updated based on these distances. While N-body simulation is also iterative, it has several differences compared with K-means algorithm: 1) N-body simulation has the same dataset (particles) for source and target set, whereas K-means operates on the different source (point) and target (cluster) sets; 2) All points in the N-body simulation would change their positions according to the time variation, whereas in K-means only the target set (cluster) would change their positions during the center update; 3) N-body simulation has the same size of source and target set, whereas K-means target set (cluster) is much smaller than source set (point) in general. We describe N-body simulation in DDSL as follow:

```

STPAcc_Iter(S){
  S = false;
  STPAcc_Comp_Dist(pSet, pSet, distMat,
                  D, "unwegL1", null);
  STPAcc_Dist_Select(distMat, idPtr, idArr,
                    range, null);
  STPAcc_Update(pSet, pSet, idPtr, idArr, S);
}

```

where range defines the value of distance threshold for selecting the neighbors.

IV. STRUCTURAL TI-BASED PRUNING

This section explains our novel algorithm optimizations, *structural TI-based pruning (STP)*, tailored for CPU-FPGA platforms. Further, we introduce an algorithm optimization flow to select the appropriate STP optimization for input algorithms. We will first introduce the standard point-based TI optimization and then illustrate our novel STP optimization.

A. Standard Point-based TI Optimization

As a simple but powerful mathematical concept, TI has been used to optimize the distance-related algorithm. Figure 2a gives an illustration. It states that $d(A, B) \leq d(A, L_{ref}) + d(L_{ref}, B)$, where $d(A, B)$ represents the distance between the point A and B in some metrics (e.g., Euclidean distance). The assistant point L_{ref} is a landmark point for reference. Directly from the definition, we could compute both the lower bound $lb(A, B)$ and upper bound $ub(A, B)$ of the distance between two points A and B . This is the standard and the most common usage of TI for deriving bounds of distances (a.k.a. the “One-landmark” bound computation).

In general, bounds can be used as a substitute for the exact distances in the distance-related data analysis. Take N-body simulation as an example. It requires to find target points that are within R (the radius) from each given query point. Suppose we get the $lb(A, B) = 10$ and $10 > R$, then we are 100% confident that source point A is not within R of query point B . As a result, it is safe to eliminate the exact distance computation between point A and B . Otherwise, the exact distance computation will still be carried out for direct comparison.

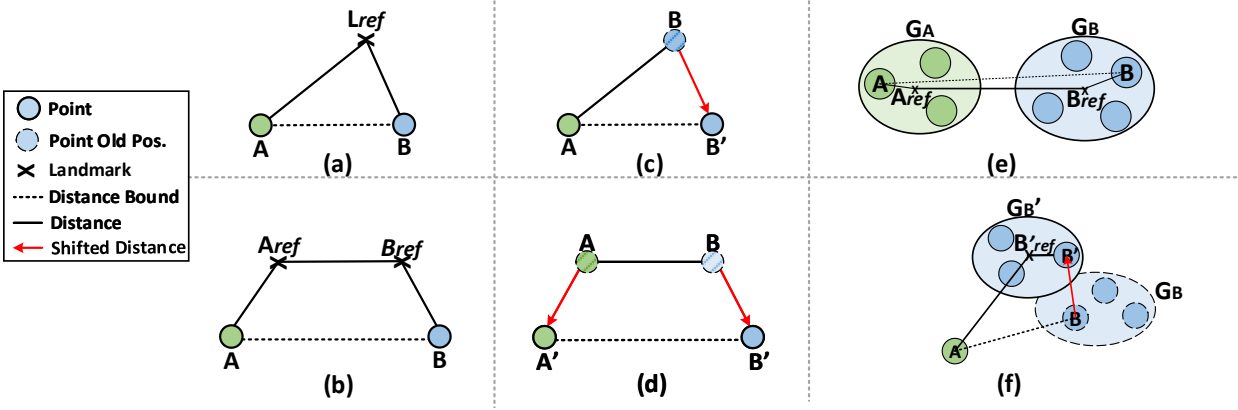


Fig. 2: Structural TI-based pruning.

B. Structural TI-based Pruning (STP)

STPAcc highlights the usage of STP by generalizing the traditional point-to-point bound computation to *two-landmark* and *trace-based* bound computation for various distance-related algorithms. Further, STP breaks the major performance hurdles of hardware acceleration of TI-optimized algorithms by combining *group-level* bound with the above two-landmark and trace-based scheme for structural pruning on redundant distance computations. Note that our STP optimization can still guarantee the same final results as the standard TI-optimized versions.

Two-landmark Bound Two-landmark bound is the first type of bound computation scheme that leverages a pair of “landmark” points to approximate the distance between two points. It aims at reducing the memory overhead of TI optimization through effective distance reuse across both source and target sets. As illustrated in Figure 2b, the distance bound between a source point A and a target B can be computed based on $d(A, A_{ref})$, $d(B, B_{ref})$ and $d(A_{ref}, B_{ref})$ through Equation 1, where A_{ref} and B_{ref} are the landmark points for point A and B in the source and target sets, correspondingly.

$$\begin{aligned} lb(A, B) &\geq d(A_{ref}, B_{ref}) - d(A, A_{ref}) - d(B, B_{ref}) \\ ub(A, B) &\leq d(A_{ref}, B_{ref}) + d(A, A_{ref}) + d(B, B_{ref}) \end{aligned} \quad (1)$$

One representative application scenario of two-landmark bound computation is KNN-join, where two disjoint sets of landmarks are selected for the source (query) and target sets.

Trace-based Bound Trace-based bound (Figure 2c) is similar to traditional point-based TI bound (Figure 2a), but differs in the selection of the reference point. It is specially crafted for iterative distance-related problems, like K-means and N-body, where the source/target point-set gets gradually updated across iterations. In these problems, the “historical” position of a point in the last iteration would generally be nice proximity of its position in the current iteration, and thus could be a nice reference point. The trace-based bound computation can work collaboratively with the aforementioned two-landmark cases for N-body simulation (as shown in Figure 2d). In N-body simulation, the source and target points are essentially the same dataset and would get updated across iterations. For simplicity, we denote the source point across iterations with A , A' , and the source point with B , B' . Clearly, we can choose

the “old” position of each point from the last iteration (A and B) as the landmark for the bound computation at the current iteration (A' and B'), due to its closeness towards the current point position. By further applying the two-landmark bound, we could then compute the bounds of distances from A' and B' in the current iteration.

Group-level Bound Group-level bound is the major driving force for our STP optimization. Group-level bound approximates the distances among different groups of points. Based on group-level bound, we can filter out distance computations for a group of points, thus, improving the computation regularity of the remaining distance computations. Note that group-level bound computation does not work by itself, but rather be combined with those point-level bound computations which we have just introduced.

As exemplified by Figure 2e and 2f, we show how group-level bound can be used accompanied by two-landmark bound (Figure 2b) and trace-based bound (Figure 2c) for structural pruning. Due to the space limit, we next give a detailed explanation of Figure 2e and leave Figure 2f to the reader. As shown in Figure 2e, we have two groups of points G_A and G_B , within which A_{ref} and B_{ref} are the shared reference points (*i.e.*, landmarks). Then based on $d(A_{ref}, B_{ref})$ and the distance between the farthest point within each group and its group reference point, $\max_{A \in G_A} d(A, A_{ref})$ and $\max_{B \in G_B} d(B, B_{ref})$, we can get the group-level bound based on Equation 2

$$\begin{aligned} lb(G_A, G_B) &\geq d(A_{ref}, B_{ref}) - \max_{A \in G_A} d(A, A_{ref}) - \max_{B \in G_B} d(B, B_{ref}) \\ ub(G_A, G_B) &\leq d(A_{ref}, B_{ref}) + \max_{A \in G_A} d(A, A_{ref}) + \max_{B \in G_B} d(B, B_{ref}) \end{aligned} \quad (2)$$

With these group-level bounds, distance computation could be filtered out at a relatively coarse granularity for structural pruning and ensure the regularity of the remaining distance computations to a larger extent. Take KNN-join as an example (Figure 2e), suppose G_A is a query group, while G_B and G_C are two target groups. If we find that $lb(G_A, G_B) > ub(G_A, G_C)$ holds and there are already more than K points in G_C , then we could safely rule out all points in G_B , as they can never be in the Top- K nearest neighbors for any point in G_A . The actual filtering scheme is more complicated, but the high-level spirit is the same. Since points inside the same source group will always maintain the same groups of target points

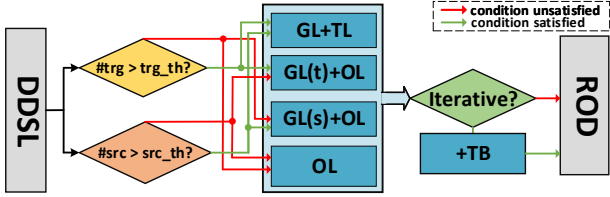


Fig. 3: Overview of the STP algorithm optimization workflow.

for distance computations after filtering, such a commonality in computation would help facilitate efficient parallelization and acceleration on the underlying hardware.

To build the initial groups, we run standard K-mean for a few (in general 5) iterations. If the input data do not have an obvious group pattern, the distance-bounds would become less effective for filtering, and more distance computations would be kept in such case. Moreover, based on our study, the cluster group center is shifted less drastically compared with the individual cluster center. Even though it may change across different iterations, it can still remain effective after several iterations in the settings of K-means and NBody. For KNN, group-level bound would remain effective since there is only one iteration.

C. Algorithm Optimization Workflow

To cover various algorithms and datasets without losing problem-specific optimization opportunities, STPAcc first decides how various STP optimizations could be applied based on the DDSL program source provided by users (as shown in Figure 3). Note that some optimization parameters, *e.g.*, the number of groups, would be decided later when more accurate information about the dataset and underlying hardware is known. The output of this optimization phase thus is just a raw-optimized design (ROD), which would then be handed over to the next phase of hardware optimization (Section VI). Specifically, STPAcc first processes the DDSL code and extracts the key meta-information for applying STP optimizations. Three key information would include the number of the source points ($\#src$), the number of the target points ($\#trg$), and the iteration property of the algorithm (identified by the `STPAcc.Iter` construct). If the size of the target points is larger than a given threshold, STPAcc framework will choose to apply the grouping on the target points. Similarly, STPAcc would also apply grouping on the source point if the size of the source point is larger than a given threshold. The threshold for the source and target points are selected based on the profiling of the distance computation filtering performance across different datasets.

STPAcc incorporates a two-phases strategy to apply STP optimization on an input algorithm, as illustrated in Figure 3. In the first phase, STPAcc decides whether to leverage group-level bound and two-landmark bound optimization based on the size of the source and target point-set. Specifically, STPAcc provides four types of optimizations, including the **Group-level (GL) + Two-landmark (TL)** bound ($\#trg > trg_th, \#src > src_th$), **Group-level (GL)(t) + One-landmark (OL)** bound ($\#trg > trg_th, \#src < src_th$), **Group-level (GL)(s) + One-landmark (OL)** bound ($\#trg <$

$trg_th, \#src > src_th$), and **One-landmark (OL)** bound ($\#trg < trg_th, \#src < src_th$). Note that Group-level (GL)(t) means applying GL on target point-set, while Group-level (GL)(s) for GL on source point-set. At the second phase, STPAcc will combine **Trace-based (TB)** bound computation if the users' algorithm contains iteration, such that more information (*e.g.*, distance drifting) can be leveraged to remove redundant distance computations. By using such a two-phase strategy, STPAcc can match a diverse range of input algorithms with more appropriate STP optimizations that can benefit their overall design performance.

V. COARSE-GRAINED WORKLOAD PARTITIONING AND MAPPING

STPAcc design is built on the CPU-FPGA architecture, which highlights its significant performance and energy efficiency, and has been widely adopted as the modern data center solution for high-performance computing and acceleration. To support the algorithm-level STP optimization on the CPU-FPGA architecture, STPAcc leverages a coarse-grained workload partitioning and mapping (WPM) strategy to manage two major types of workloads, where the first type (**Type I**) is composed of control-intensive tasks, including the distance-bound computation, distance filtering operations and data regrouping. And the second type (**Type II**) consists of compute-intensive workloads for the initial point clustering and the remaining distance computations.

The operations in Type I workload are complex in their execution flow (irregular, with much data dependency). Therefore, we allocate them to the CPU for sequential execution. The operations in Type II workload, on the other hand, consist of distance computations with simple execution flow (regular, with little data dependency). Thus, we assign them to the FPGA for acceleration due to their potential for more fine-grained data-level parallelization and computation pipelining.

Depending on the input data and the underlying hardware properties, the ratio of Type I and Type II workload can be adjusted to meet the performance requirements and hardware constraints. STPAcc introduces one major parameter – the number of groups, to control such ratio, as shown in Figure 4. In short, with more cluster groups, STP optimization would remove more distance computations due to its more fine-grained filtering, thus, leading to more Type I workload at CPU side and fewer Type II workloads (distance computations) executed on the FPGA. Conversely, with a fewer number of cluster groups, STP optimization would spot and remove fewer redundant distance computations because of lacking sufficient bounds for fine-grained filtering, therefore, leading to fewer filtering computations (Type I workload) on the CPU side, but more remaining distance computations (Type II workload) on the FPGA. To determine the optimal value of the number of groups, we leverage a light-weighted ML model (Regressor) to select the appropriate number of groups given the input dataset size and dimensionality, hardware compute and memory resource, and the estimated filtering performance. Note that we estimate filtering performance of a given group number through down-sampling based profiling

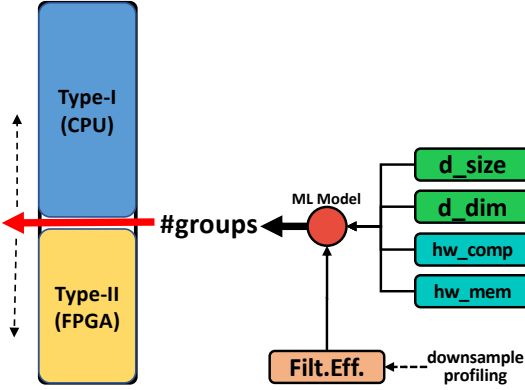


Fig. 4: Workload partitioning and mapping based on the number of groups.

on the incoming datasets by running K-means on sampled small datasets for few iterations.

One more thing to note is that the operations such as the distance bounds computations and triangle-inequality-based filtering, which are irregular (lots of condition checking and execution dependency) are suitable for CPUs. Compared with CPU-based computation, the data transfer overhead is largely minor. Based on our profiling, Based on the profiling of our evaluated datasets, the FPGA side workload (after the filtering) would take an 83% running time on average, which dominates the overall execution time. Therefore, putting those distance computation workloads on FPGAs, we can still benefit from the high pipeline and parallel execution compared with CPUs with limited parallelization capability.

VI. FINE-GRAINED HARDWARE OPTIMIZATIONS

While FPGA accelerator features with high computation capability, the memory bandwidth bottleneck constraints the overall design performance. Therefore, optimizing data placement and memory architecture is the key to improving memory performance. In addition, the OpenCL-based programming model adds a layer of architectural complexity of the kernel design and management, which is also critical to the design performance. STPAcc framework distinguishes itself by using the novel memory and kernel optimization strategies that are tailored for STP-optimized distance-related algorithms to benefit CPU-FPGA designs.

A. Memory Optimization

After applying the STP optimization to remove the redundant distance computation, each source point group will have different target groups as candidates for distance computation, as shown in Figure 5a, where **Source-grp** is the ID of the source group, and **Target-grp** is ID of the target group. However, this would raise two performance concerns.

Inter-group memory irregularity It would lead to low data reuse (Figure 5a). For example, the target group information (t_1, t_4, t_6) required by source group s_1 can not be reused by s_2 . Since s_2 requires quite different target groups (t_8, t_{10} , and t_{12}) for distance computation, thus, additional costly memory access has to be carried out. To tackle this problem, STPAcc places the source groups to the continuous memory

Source-grp	Target-grp	Source-grp	Target-grp
s_1	t_1, t_4, t_6	s_1	t_2, t_4, t_6
s_2	t_8, t_{10}, t_{12}	s_5	t_2, t_4, t_6
...	...	s_2	t_8, t_{10}, t_{12}
s_5	t_2, t_4, t_6	s_6	t_8, t_{10}, t_{12}
s_6	t_8, t_{10}, t_{12}

Fig. 5: (a) Non-optimized inter-group memory access; (b) Optimized inter-group memory access.

space to maximize the memory access efficiency, only if these source groups have the same set of target groups as candidates for distance computation. An example has been shown in Figure 5b, where the source group s_2 and s_6 are placed side by side in the memory since they have the same list of target groups (t_8, t_{10} , and t_{12}), which can take advantage of the memory temporal locality without additional memory access.

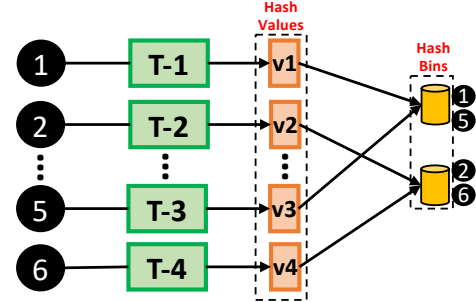


Fig. 6: Point-group clustering.

To fast organize the source groups, we adopt a locality-sensitive hashing clustering strategy that operates on the source groups. as described in Figure 6, we treat the target groups of each source group as an “attribute” of the source group, which is essentially a group id-list. We apply Locality-Sensitive Hashing, which has been widely applied in grouping similar items [6], [7], [13], to cluster the source groups based on their list of target groups, where the source group with similar target group lists will be more likely put to the same hash bin (cluster). Such a hash-based solution is lightweight with $O(m)$ time complexity, where m is the number of source groups.

Intra-group memory irregularity It is caused by points grouping in STP optimization and would result in inefficient memory access. For example, points from *group 1, 2, and 3* have taken up the memory space at intervals, as shown in Figure 7a. However, a group of points are usually accessed simultaneously due to STP optimization. This would cause frequent inefficient memory access for fetching individual points distributed at the discontinuous memory address, as shown in Figure 7b. To solve this issue, STPAcc offers a second memory optimization — *intra-group node reordering*, to reorganize the target/source points inside the same target/source group into continuous memory space within the same memory bank, as illustrated in Figure 7c. This strategy can largely benefit memory coalescing and external memory bandwidth while minimizing the access contention, since points inside the same bank can be accessed efficiently and points inside different banks can be accessed in parallel.

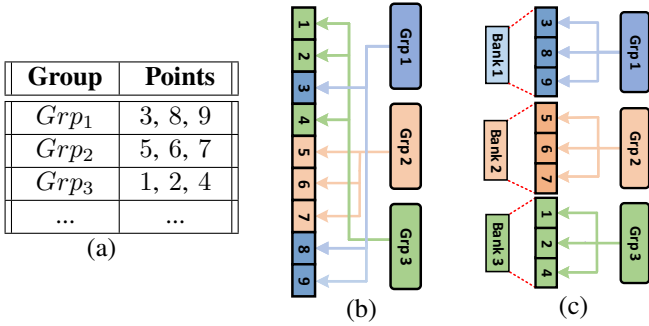


Fig. 7: (a) Group-point mapping; (b) Non-aligned intra-group memory; (c) Aligned intra-group memory.

B. OpenCL Kernel Design

To process the remaining distance computation on the FPGA, STPacc customizes OpenCL kernel design in support of STP optimization. Specifically, for each of those distance algorithms, our first step is to do structural pruning for reducing the unnecessary computations on CPU host, where we can apply the aforementioned distance-bound based filtering depending on the distance application settings. Once we finish this step, the remaining distance computations that have not been filtered out will be put to FPGAs for computations. To make these computations more suitable for massive parallel FPGA computation, we leverage the memory optimization discussed in the previous subsection to make the remaining computation more regularized. This step also carried on CPU host. Then, those regularized computations workload along with their involved data (*e.g.*, points, clusters, point/cluster groups) will be handed over to FPGA for acceleration. To running the OpenCL-based FPGA kernel, we will first copy the data from host main memory to FPGA device memory (get the device memory address pointers) and then call the OpenCL kernel on CPU host for processing the remaining distance computations on FPGAs.

The detailed design of our OpenCL-based FPGA kernel is shown in Figure 8, which consists of a set of parallel workload queues and parallel kernels. Specifically, parallel workload queues is the place to hold the distance computation between source and target groups, while the parallel kernels would fetch the workload from their corresponding queue and load it to global memory, then do GEMM-based point-to-point computation by leverage compute units of the FPGA. We evenly distribute workload based on the number of parallel workload queues and the total number workloads considering that 1) each kernel is short in its execution time and 2) the number of parallel queues is much smaller than the total number of workloads. Even though more complicated schedule strategies exist, such as work stealing that can put all active kernels fully in use, it will inevitably lead to the additional time and computation cost that would hurt performance.

VII. EVALUATION

To show the performance and energy efficiency advantage of the proposed STPacc framework, we conduct experiments over various benchmarks, datasets, and hardware platforms.

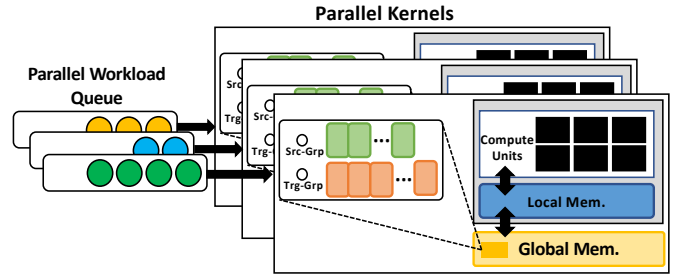


Fig. 8: OpenCL kernel design.

TABLE II: CPU-based and STPacc Implementation.

Name	Techniques	Description
Sequential (Baseline)	Standard Algorithm without any optimization (CPU).	Naive for-loop based implementation on CPU.
TI-opt	Point-based TI-Optimized Algorithms (CPU).	TI-optimized distance-related algorithm [22], [61], [18] running on CPU.
CBLAS	CBLAS library Accelerated Algorithms (CPU).	Standard distance-related algorithm with CBLAS [60] acceleration.
STPacc	Algorithmic-hardware co-design (CPU-FPGA).	STP optimization and FPGA acceleration of distance computations.

A. Experiment Setup

Benchmarks We choose three representative benchmarks (K-means, KNN-join, and N-body Simulation) and cover popular datasets from machine learning and algorithm optimization domains. The major reason of selecting these three algorithms is to cover the major types of computation patterns that are widely existed in different distance-related algorithms. Specifically, **K-means** can show the benefits of STPacc hierarchy (*One-landmark + Trace-based + Group-level*) bound computation optimization on the iterative algorithms with disjoint source and target set. **KNN-join** would demonstrate the effectiveness of STPacc hybrid (*Two-landmark + Group-level*) bound computation optimization on the non-iterative algorithms. **N-body Simulation** would show the strength of STPacc hybrid bound computation (*Two-landmark + Trace-based + Group-level*) on iterative algorithms with the same source and target set.

Datasets In the evaluation, we use six datasets for each algorithm. The selected datasets can cover a wide spectrum of mainstream datasets, including datasets from UCI Machine Learning Repository [21], and datasets that have been used by previous papers [18], [19], [9] in the related domains. Details of these datasets are listed in Table III. Note that the KNN-join algorithm will find the Top-1,000 closest neighbors of each query point.

Baselines We build implementations for CPU, FPGA, and GPU platforms for evaluation and comparison. **CPU-based implementations.** We include three CPU-based implementations for all benchmarks to give a comprehensive demonstration of the benefits of our STPacc (summarized in the

TABLE III: Datasets for evaluation.(#G is the number of groups determined by our ML model.)

K-means				KNN-join				N-body Simulation			
Dataset	Size	Dim.	#Cluster	#G	Dataset	Dim.	#Source	#G	Dataset	#Particle	#G
Poker Hand	25,010	11	158	13	Harddrive1	64	68,411	785	P-1	16,384	384
Smartwatch Sens	58,371	12	242	16	Kegg Net Directed	24	53,413	687	P-2	32,768	543
Healthy Older People	75,128	9	274	17	3D Spatial Network	3	434,874	1,978	P-3	59,049	729
KDD Cup 2004	285,409	74	534	23	KDD Cup 1998	56	95,413	927	P-4	78,125	839
Kegg Net Undirected	65,554	28	256	16	Skin NonSkin	4	245,057	1,485	P-5	177,147	1,263
Ipums	70,187	60	265	15	Protein	11	26,611	489	P-6	262,144	1,536

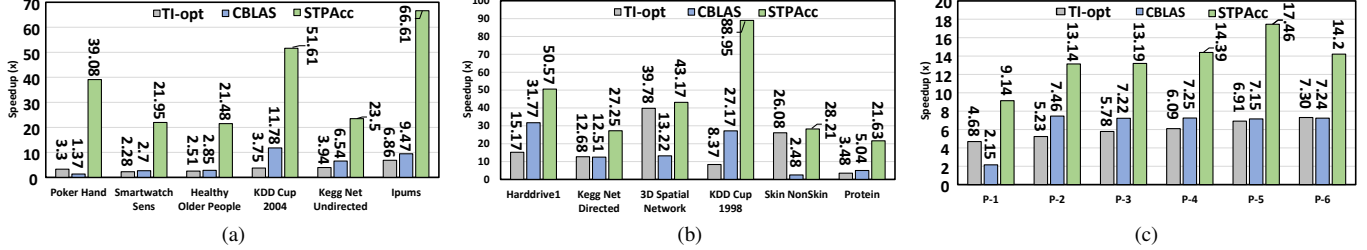


Fig. 9: Performance comparison (TI-opt, CBLAS, STPAcc): (a) K-means (b) KNN-Join (c) N-body Simulation.

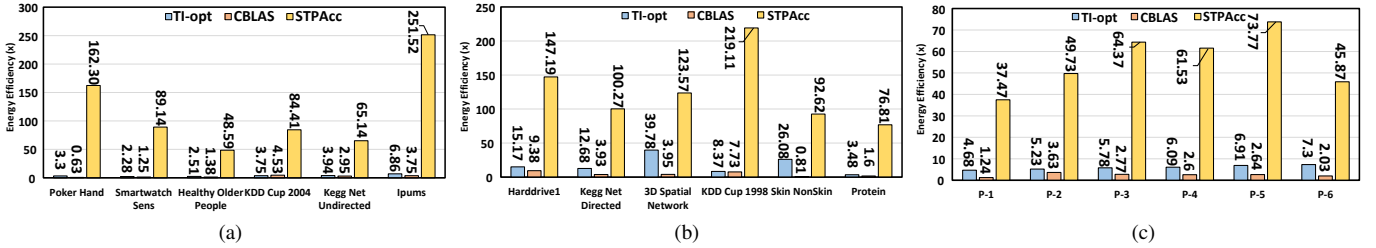


Fig. 10: Energy efficiency comparison (TI-opt, CBLAS, STPAcc): (a) K-means. (b) KNN-Join. (c) N-body Simulation.

first three rows in Table III: 1) the *naive for-loop sequential* code without any optimization (also used to normalize the speedup and energy-efficiency of other implementations), 2) previous *TI-based* implementations [22], [61], [18], 3) the *CBLAS-based* [60] implementations. **FPGA-based Implementations.** It is challenging to give a fair comparison with other FPGA-based implementations since existing FPGA designs are mostly NOT open-sourced and are ONLY customized for a specific algorithm/input/hardware. We instead use K-means as a showcase to provide some deeper understanding of our algorithm-hardware co-design advantages. **GPU-based Implementations.** Different from FPGA-based implementations, which aim at better Pareto-optimal solutions in terms of both performance and energy efficiency, GPUs mostly offer better performance at a cost of much higher energy consumption. To further demonstrate the benefits of using CPU-FPGA platforms, we compare our STPAcc with a manually-optimized implementation of the standard KNN-join algorithm on the GPU. Note that the naive for-loop baseline is used to normalize the speedup and analyze performance benefits. Other CPU-based (TI-opt and CBLAS) and GPU-based implementations are the optimized multi-threaded implementations based on floating-point computation that can represent the state-of-the-art performance of these algorithms on CPUs and GPUs.

Platforms We use Intel Stratix 10 GX DE10-Pro [3] as the FPGA accelerator (running at 310MHz) and run the host side software program on Intel Xeon Silver 4110 processor [2] (8-core 16-thread, 2.1GHz base clock frequency, 85W TDP). DE10-Pro FPGA has 378,000 Logic elements (LEs), 128,160 adaptive logic modules (ALM), 512,640 ALM registers, 648 DSPs, and 1,537 M20K memory blocks. We im-

plement STPAcc designs on DE10-Pro by using Intel Quartus Prime Software Suite [4] with Intel FPGA OpenCL SDK included. For GPU-comparison, we use NVIDIA Quadro P6000 GPU [48] (3840 CUDA cores, Memory: 24GB GDDR5X, Peak Memory Bandwidth: 432GB/s, Peak Single Precision Performance: 12 TFLOPs). Note that for GPU and FPGA designs, we use PCIe 3.0 x16 for data transferring between the host CPU and GPU/FPGA.

B. Results

In this section, we compare STPAcc with CPU-based implementations. We first demonstrate the advantage of STPAcc in terms of performance and energy efficiency. Then we conduct a detailed performance benefits breakdown on K-means to demonstrate the effectiveness of our STP algorithmic optimization and hardware design on the CPU-FPGA platform.

Performance As shown in Figure 9, STPAcc, TI-opt, and CBLAS achieve $31.42\times$, $9.12\times$, and $9.19\times$ speedup on average compared to the naive for-loop CPU baselines, respectively. STPAcc outperforms all three CPU-based implementations across all benchmarks and datasets of various sizes and dimensions, especially on those inputs of large sizes and high dimensionality. For example, on datasets KDD Cup 2004 ($n = 285,409, d = 74$) and Ipums ($n = 70,187, d = 60$) for K-means, STPAcc achieves $51.61\times$ and $66.61\times$ speedup over our baseline, and also significantly higher than both TI-opt and CBLAS implementations. A similar observation can also be concluded from KNN-join, such as $88.95\times$ speedup on dataset KDD Cup 1998 ($n = 95,413, d = 56$). The reason for such performance improvement is that our STPAcc design

TABLE IV: Comparison with the state-of-the-art.

Dimension	[40] (ms)	STPAcc (ms)	Speedup
54	58.68	21.60	2.72×
78	45.36	23.20	1.96×
128	158.12	24.09	6.56×

can effectively reconcile the benefits from both the STP optimization and the FPGA acceleration, where the former provides the opportunity to reduce the distance computation at the algorithm level (benefiting more on large datasets), and the latter boosts the performance from hardware acceleration perspective (more suitable for high-dimensional cases).

Energy efficiency The energy efficiency of STPAcc design is also significant. Figure 10 shows the energy efficiency of STPAcc, TI-opt, and CBLAS normalized to the baseline. STPAcc designs deliver an average $116.85\times$ better energy efficiency compared with the baseline, which is significantly higher than TI-opt and CBLAS implementations. There are two reasons behind these results: 1) much lower power consumption. STPAcc CPU-FPGA design only consumes 5 Watt to 17.12 Watt across all algorithm and dataset settings, whereas Intel Xeon CPU consumes at least 20.9 Watt and 42.49 Watt on TI-opt and CBLAS implementations, respectively; 2) considerable performance. STPAcc design achieves a much better speedup (more than $3\times$ on average) compared with the TI-opt and CBLAS, which contributes to the overall design energy-efficiency. Among these implementations, CBLAS implementation has the lowest energy efficiency, since it relies on multi-core parallel processing capability of the CPU, which improves the performance at the cost of much higher power consumption (average 65.79 Watt). TI-opt only leverages the single-core processing capability of the CPU and achieves moderate performance with effective distance computation reduction, which results in less power consumption (average 25.59 Watt) and higher energy efficiency (average $9.12\times$) compared with the baseline. Different from the TI-opt and CBLAS implementations, STPAcc design is built upon a low-power platform with considerable performance, which shows a far better energy-performance trade-off.

Compared with the state-of-the-art we also compare our STPAcc design with the state-of-the-art OpenCL-based CPU-FPGA design from [40] on KNN algorithm. [40] evaluates its designs on the dataset with 20,480 samples, 20 clusters with three different dimension sizes 64, 78, 128. And [40] uses the same compilation tool and platforms as ours: 1) Intel OpenCL SDK for FPGA; 2) Intel Stratix series accelerator card. As shown in Table IV, different dimension settings, our STPAcc can consistently offer performance speedup with an average of $3.75\times$. The major reason behind this is that [40] only incorporate a Bitonic sort to improve the neighbor distance ordering process, which is largely minor compared with the high-overhead distance computation process. In contrast, our STPAcc not only includes the TI-based distance filtering algorithmic innovation to reduce the unnecessary high-cost distance computations but also tailors FPGA hardware level design to exploit such benefits effectively.

Performance-benefits Analysis on K-means To show the

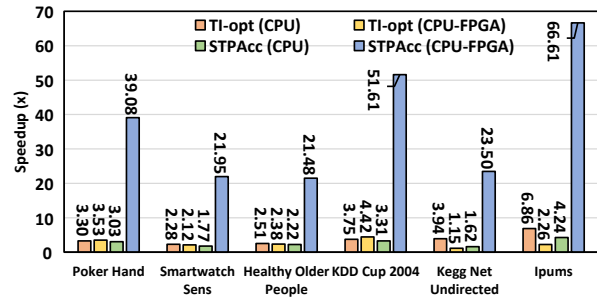


Fig. 11: Detailed performance comparison on K-means.

performance-gain breakdown of STPAcc, we built four different implementations for K-means: 1) TI-opt K-means on the CPU; 2) TI-opt K-means on the CPU-FPGA platform; 3) STPAcc K-means on the CPU; 4) STPAcc K-means on the CPU-FPGA platform. Note that TI-opt K-means is designed for sequential-based CPUs and there is no publicly available TI-opt implementation on CPU-FPGA platforms. For a fair comparison, we implement TI-opt K-means on the CPU-FPGA platform with memory optimizations (inter-group and intra-group memory optimization) and distance computation kernel optimization (Vector-Matrix multiplication). We normalize the speedup performance of each implementation w.r.t. the naive for-loop K-means on CPU. Note that for CPU-based STPAcc, we just process the workload of the remaining distance computations (after the distance filtering) on the CPU instead of offloading to FPGAs. Given the same workload, the WPM framework would generate the same ratio between Type-I and Type II workload for both CPU-based STPAcc and FPGA-based STPAcc. Since our STPAcc design is mainly optimized for FPGA-based implementations, where CPU-based design is chosen as our baseline for comparison. As shown in Figure 11, TI-opt K-means (CPU) achieves an average $3.77\times$ speedup, whereas TI-opt K-means (CPU-FPGA) only achieves an average $2.63\times$ speedup. Although we manage to add possible optimizations in TI-opt (CPU-FPGA), applying fine-grained TI optimization would still cause a large divergence of computation among points, leading to low data reuse and inefficient memory access. We also notice that the STP optimization succeeds to achieve good computation/memory access regularity for better hardware acceleration while maintaining similar power of computation elimination compared with the standard TI optimization. Note that STPAcc (CPU) could lead to lower speedup (average $2.69\times$) compared with the TI-opt (CPU) (average $3.77\times$), since STPAcc leverages coarse-grained STP optimization that spots fewer number of unnecessary distance computations, which is the major source of performance improvements on the CPU. However, when combining STPAcc design with the CPU-FPGA platform, the benefits of STPAcc STP optimization become prominent (average $37.37\times$), since it can maintain computation regularity while reducing memory overhead to facilitate the hardware acceleration on the FPGA. Whereas, TI-opt (CPU-FPGA) strikes to maximize the algorithm-level benefits while ignoring hardware-level properties, leading to poor performance.

STPAcc vs. GPU We also compare STPAcc with GPU-based implementations on all three algorithms (K-means,

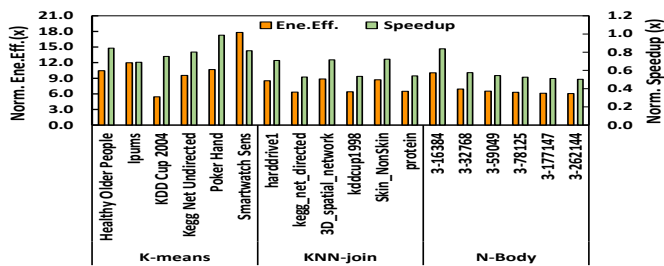


Fig. 12: Speedup and energy-efficiency vs. GPU.

KNN-join, and N-body) and observe a similar pattern in the results. Note that GPU-based implementation is sharing the same OpenCL code as our FPGA implementation. As shown in Figure 12, we observe STPAcc achieves consistently better energy efficiency performance ($5.42\times \sim 17.81\times$, with average $8.51\times$) compared with GPU across all datasets on three algorithms. Although GPU is faster in its speedup performance due to its massive number of CUDA cores that can easily launch more threads for parallelization, it also comes with a huge cost of power consumption (average 85.72 Watt) compared with STPAcc (7.14 Watt). This comparison also highlights the potential of STPAcc for energy saving and application in power-constraints settings.

VIII. CONCLUSION

In this paper, we present our STPAcc framework to accelerate the distance-related algorithms on the CPU-FPGA platform. Specifically, STPAcc leverages a simple but expressive language construct (DDSL) to unify the distance-related algorithms, and a comprehensive optimizing strategy to improve the design performance from algorithmic and hardware perspective systematically and automatically. Rigorous experiments on three popular algorithms (K-means, KNN-join, and N-body simulation) demonstrate the STPAcc as a powerful and comprehensive framework for hardware acceleration of distance-related algorithms on modern CPU-FPGA platforms.

REFERENCES

- [1] Intel quartus prime software suite. <https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html>.
- [2] Intel xeon silver 4110 processor.
- [3] Terasic de10-pro stratix 10 gx/sx fpga development kit.
- [4] D.A. Adeniyi, Zune Wai, and Y. Yongquan. Automated web usage data mining and recommendation system using k-nearest neighbor (knn) classification method. *Applied Computing and Informatics*, 36, 10 2014.
- [5] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 1992.
- [6] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*, pages 459–468. IEEE, 2006.
- [7] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. In *Advances in neural information processing systems*, pages 1225–1233, 2015.
- [8] J. Canilho, M. Véstias, and H. Neto. Multi-core for k-means clustering on fpga. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016.
- [9] Guoyang Chen, Yufei Ding, and Xipeng Shen. Sweet knn: An efficient knn on gpu through reconciliation between redundancy removal and regularity. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, 2017.
- [10] Yang Chen and Gérard Medioni. Object modeling by registration of multiple range images. In *Robotics and Automation, IEEE*, pages 2724–2729, 1991.
- [11] Adam Coates and Andrew Y Ng. Learning feature representations with k-means. In *Neural networks: Tricks of the trade*. 2012.
- [12] National Research Council. *Frontiers in massive data analysis*. National Academies Press, 2013.
- [13] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.
- [14] Cheng-Hao Deng and Wan-Lei Zhao. Fast k-means based on k-nn graph. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1220–1223. IEEE, 2018.
- [15] Nameirakpam Dhanachandra, Khumanthem Manglem, and Yambem Jina Chanu. Image segmentation using k-means clustering algorithm and subtractive clustering algorithm. *Procedia Computer Science*, 54:764–771, 2015.
- [16] G. Di Fatta and D. Pettinger. Dynamic load balancing in parallel kd-tree k-means. In *2010 10th IEEE International Conference on Computer and Information Technology (ICCIT)*, 2010.
- [17] Edsger W Dijkstra. A note on two problems in connexion with graphs. In *Numerische mathematik*, volume 1, pages 269–271, 1959.
- [18] Yufei Ding, Xipeng Shen, Madanlal Musuvathi, and Todd Mytkowicz. Top: A framework for enabling algorithmic optimizations for distance-related problems. *Proc. VLDB Endow.*, 2015.
- [19] Yufei Ding, Yue Zhao, Xipeng Shen, Madanlal Musuvathi, and Todd Mytkowicz. Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup. In *International Conference on Machine Learning (ICML)*, 2015.
- [20] Jonathan Drake and Greg Hamerly. Accelerated k-means with adaptive distance bounds.
- [21] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.
- [22] Charles Elkan. Using the triangle inequality to accelerate k-means. In *Proceedings of the 20th International Conference on Machine Learning (ICML)*, 2003.
- [23] AM Fahim, AM Salem, F Af Torkey, and MA Ramadan. An efficient enhanced k-means clustering algorithm. *Journal of Zhejiang University-Science A*, 7(10):1626–1633, 2006.
- [24] Miquel Ferrer, Ernest Valveny, Francesc Serratos, Itziar Bardaji, and Horst Bunke. Graph-based k-means clustering: A comparison of the set median versus the generalized median graph. In *International Conference on Computer Analysis of Images and Patterns*, pages 342–350. Springer, 2009.
- [25] Evelyn Fix and Joseph L Hodges Jr. Discriminatory analysis-nonparametric discrimination: consistency properties. In *DTIC Document*.
- [26] Michael Gowanlock. Knn-joins using a hybrid approach: Exploiting cpu/gpu workload characteristics. In *Proceedings of the 12th Workshop on General Purpose Processing Using GPUs (GPGPU)*, 2019.
- [27] Greg Hamerly. Making k-means even faster. In *Proceedings of the 2010 SIAM international conference on data mining*, pages 130–140. SIAM, 2010.
- [28] H. Hussain, K. Benkrid, C. Hong, and H. Seker. An adaptive fpga implementation of multi-core k-nearest neighbour ensemble classifier using dynamic partial reconfiguration. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, 2012.
- [29] H. M. Hussain, K. Benkrid, A. T. Erdogan, and H. Seker. Highly parameterized k-means clustering on fpgas: Comparative results with gpps and gpus. In *2011 International Conference on Reconfigurable Computing and FPGAs*, 2011.
- [30] H. M. Hussain, K. Benkrid, and H. Seker. An adaptive implementation of a dynamically reconfigurable k-nearest neighbour classifier on fpga. In *2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2012.
- [31] H. M. Hussain, K. Benkrid, H. Seker, and A. T. Erdogan. Fpga implementation of k-means algorithm for bioinformatics application: An accelerated approach to clustering microarray data. In *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2011.
- [32] Shigeru Ida and Junichiro Makino. N-body simulation of gravitational interaction between planetesimals and a protoplanet: I. velocity distribution of planetesimals. 1992.
- [33] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 2010.
- [34] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. An efficient k-means

- clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis & Machine Intelligence (PAMI)*, 2002.
- [35] Dominique Lavenier. Fpga implementation of the k-means clustering algorithm for hyperspectral images. *Los Alamos National Laboratory LAUR*, 2000.
- [36] Chao Li, Shuheng Zhang, Huan Zhang, Lifang Pang, Kinman Lam, Chun Hui, and Su Zhang. Using the k-nearest neighbor algorithm for the classification of lymph node metastasis in gastric cancer. *Computational and mathematical methods in medicine*, 2012, 2012.
- [37] Youguo Li and Haiyan Wu. A clustering method based on k-means algorithm. *Physics Procedia*, 25:1104–1109, 2012.
- [38] Zhe-Hao Li, Ji-Fang Jin, Xue-Gong Zhou, and Zhi-Hua Feng. K-nearest neighbor algorithm implementation on fpga using high level synthesis. In *2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, 2016.
- [39] Zhongduo Lin, Charles Lo, and Paul Chow. K-means implementation on fpga for high-dimensional data using triangle inequality. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, 2012.
- [40] Liyuan Liu and Mohammed AS Khalid. Acceleration of k-nearest neighbor algorithm on fpga using intel sdk for opencl. In *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1070–1073. IEEE, 2018.
- [41] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yanzhi Wang, Jian Tang, and Jieping Ye. Autocompress: An automatic DNN structured pruning framework for ultra-high compression rates. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 4876–4883. AAAI Press, 2020.
- [42] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 1982.
- [43] Xiaolong Ma, Fu-Ming Guo, Wei Niu, Xue Lin, Jian Tang, Kaisheng Ma, Bin Ren, and Yanzhi Wang. Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices.
- [44] E. S. Manolakos and I. Stamoulias. Ip-cores design for the knn classifier. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 2010.
- [45] Harshitha Menon, Lukasz Wesolowski, Gengbin Zheng, Pritish Jetley, Laxmikant Kale, Thomas Quinn, and Fabio Governato. Adaptive techniques for clustered n-body cosmological simulations. *Computational Astrophysics and Cosmology*, 2(1):1–16, 2015.
- [46] Wang Kay Ngai, Ben Kao, Chun Kit Chui, Reynold Cheng, Michael Chau, and Kevin Y Yip. Efficient clustering of uncertain data. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 436–445. IEEE, 2006.
- [47] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, ASPLOS '20, page 907–922, New York, NY, USA, 2020. Association for Computing Machinery.
- [48] Nvidia. Quadro p6000. www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/productspage/quadro/quadro-desktop/quadro-pascal-p6000-data-sheet-us-nv-704590-r1.pdf
- [49] Lars Nyrons. Fast n-body simulation with cuda. 2007.
- [50] Yiwei Pan, Zhibin Pan, Yikun Wang, and Wei Wang. A new fast search algorithm for exact k-nearest neighbors based on optimal triangle-inequality-based check strategy. *Knowl. Based Syst.*, 189, 2020.
- [51] Jianpeng Qi, Yanwei Yu, Lihong Wang, Jinglei Liu, and Yingjie Wang. An effective and efficient hierarchical k-means clustering algorithm. *International Journal of Distributed Sensor Networks*, 13(8):1550147717728627, 2017.
- [52] Siddheswar Ray and Rose H Turi. Determination of number of clusters in k-means clustering and application in colour image segmentation. In *Proceedings of the 4th international conference on advances in pattern recognition and digital techniques*, 1999.
- [53] T. Saegusa and T. Maruyama. An fpga implementation of k-means clustering for color images based on kd-tree. In *2006 International Conference on Field Programmable Logic and Applications (FPL)*.
- [54] D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web, WWW '10*, pages 1177–1178, New York, NY, USA, 2010. ACM.
- [55] Rainer Spurzem. Direct n-body simulations. *Journal of Computational and Applied Mathematics*, 109(1-2):407–432, 1999.
- [56] Ioannis Stamoulias and Elias S. Manolakos. Parallel architectures for the knn classifier – design of soft ip cores and fpga implementations. *ACM Trans. Embed. Comput. Syst.*, 2013.
- [57] Miren Tian, Xin'an Wang, Xing Zhang, Zhiqiang Yang, Jipan Huang, and Hao Chen. The implementation of a knn classifier on fpga with a parallel and pipelined architecture based on predetermined range search. In *2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*.
- [58] M. Trenti and P. Hut. N-body simulations (gravitational). *Scholarpedia*.
- [59] Jingdong Wang, Jing Wang, Qifa Ke, Gang Zeng, and Shipeng Li. Fast approximate k-means via cluster closures. In *Multimedia data mining and analytics*, pages 373–395. Springer, 2015.
- [60] Qian Wang, Xianyi Zhang, Yunquan Zhang, and Qing Yi. Augem: automatically generate high performance dense linear algebra kernels on x86 cpus. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2013.
- [61] Xueyi Wang. A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1293–1299. IEEE, 2011.
- [62] Xueyi Wang. A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality. *Proceedings of International Joint Conference on Neural Networks / co-sponsored by Japanese Neural Network Society (JNNS). International Joint Conference on Neural Networks*, 43:2351–2358, 02 2012.
- [63] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.
- [64] C. Yang, X. Yu, and Y. Liu. Towards efficient knn joins on data streams. In *2014 IEEE International Congress on Big Data*, 2014.
- [65] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, ISCA '17, page 548–560, New York, NY, USA, 2017. Association for Computing Machinery.
- [66] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, 2018.