AccD: A Compiler-based Framework for Accelerating Distance-related Algorithms on CPU-FPGA Platform

Abstract

*Distance-related algorithm (e.g. KMeans, and KNN) plays an essential role across various domains (e.g. Machine Learning and Computer Network), but it comes with high computation complexity, which leads to poor performance and limited applicability. Existing optimizations of distance-related algorithm require non-trivial efforts in detailed algorithmic optimization and hardware implementation, which are time-consuming and lack of generality. To this end, we propose **AccD**, a compiler-based framework for accelerating distance-related algorithms on CPU-FPGA platform. Specifically, AccD provides a Domain-specific Language to unify distance-related algorithms effectively, and an optimizing compiler to reconcile the benefits from both algorithmic optimization on CPU and hardware acceleration on FPGA. The output of AccD is a high-performance and power-efficient design that can be easily synthesized and deployed on the mainstream CPU-FPGA platforms. Intensive experiments show that AccD designs achieve 24.17× speedup and 77.98× better energy efficiency on average over the standard CPU-based implementations on a server-grade Intel Xeon processor.*

1. Introduction

add your new version here. FPGA-based heterogeneous system (CPU-FPGA) using the OpenCL standard [1–3] has recently emerged as a promising hardware platform for program acceleration [4–6], due to its strengths in programming flexibility, significant performance, and high energy-efficiency. Yet, a general compiler framework is still missing, which would enable automatic algorithm-hardware co-optimization of an important type of computation-intensive algorithms, **distance-related algorithms**. These algorithms play a vital role in many domains, including machine learning (e.g., KMeans [7], and KNN [8]), computational physics (e.g., Nbody Simulation [9]), etc. Specifically, we found four major gaps between existing efforts and the proposed compiler framework.

Rely on problem-specific design and optimization while missing effective generalization. There is no such unified abstraction to formalize the definition and optimization of distance algorithm systematically. Most of the previous hardware designs and optimizations [10–13] are heavily coded for a specific algorithm (e.g., KMeans), which can not be shared with different distance-related algorithms. Moreover, these "hard-coded" strategies could also fail to catch up with the ever-changing upper-level algorithmic optimizations and the underlying hardware settings, which could result in large cost of re-design and re-implementation in the design evolvement.

Lack of algorithm-hardware co-design. Previous algorithmic [14, 15] and hardware optimizations [10–13, 16] are usually applied separately instead of combined collaboratively. Existing algorithmic optimizations, most of which based on *triangle inequality (TI)* [14, 15, 17, 18], are crafted for sequential-based CPU. Despite removing large amount of distance computations, they also incur high computation irregularity and memory overhead. Therefore, directly applying these algorithmic optimizations to massive parallel platform (e.g., CPU-FPGA) without taking appropriate hardware-aware adaption could even lead to inferior performance [18].

Count on FPGA as the only source of acceleration. Previous work [10, 13, 19–22] place the whole algorithm on the FPGA accelerator without considering the assists from different computing resources. As a result, their designs are usually limited by the on-chip memory and computing elements. Moreover, it cannot fully exploit the power of the FPGA while missing the potential performance benefits from the heterogeneous computing paradigm, such as using CPU for complex logic and control operations while offloading the compute-intensive task to FPGA.

Lack of well-structured design workflow. Previous work [11–13, 19, 21] follow the traditional way of hardware implementation and require intensive user involvement in hardware design, implementation, and extra manual tuning process, which usually takes long development-to-validation cycles. Also, the problem-specific strategy leads to a case-by-case design process, which cannot be widely applied to handle different problem settings.

To this end, we present an OpenCL-based compiler framework, *AccD*, to automatically accelerate distance-related algorithms on the CPU-FPGA platform (shown in Figure 1). First, AccD provides a **distance-related domain-specific language** (*DDSL*) as a problem-independent abstraction to unify the description and optimization of various distance-related algorithms. With the assist of the *DDSL*, end-user can easily create highly-efficient CPU-FPGA design by only focusing on high-level problem specification without touching the arduous algorithmic optimization or hardware implementation.

Second, AccD offers a novel **algorithmic-hardware co-optimization** scheme to reconcile the acceleration from both sides. At the algorithmic level, AccD incorporates a novel *generalized triangle inequality optimization (GTI)* to eliminate unnecessary distance computation on the CPU, while maintaining the computation regularity to a large extent. At the hardware level, AccD employs a *specialized data layout* to enforce memory coalescing and an *optimized distance computa-*

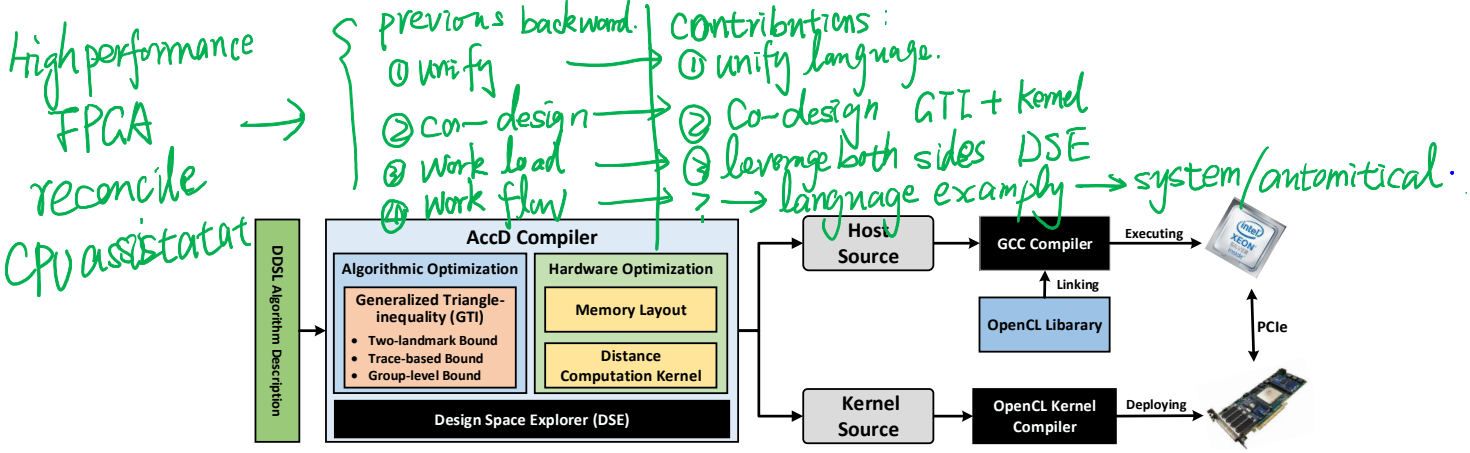


Figure 1: AccD Overview.

tion kernel to accelerate the remaining distance computations on FPGA.

Third, AccD leverages both the host and accelerator side of the **CPU-FPGA heterogeneous system for acceleration**. In particular, AccD distributes the algorithmic-level optimization (e.g., data grouping and distance computation filtering) to CPU, which consists of complex operations and execution dependency, but lack of pipeline and parallelism. On the other hand, AccD assigns the hardware-level optimization (e.g., distance computation acceleration) to the FPGA, which are composed of simple and vectorizable operations. Such simple mapping thus successfully capitalizes on the benefit of CPU for managing control-intensive tasks and the advantage of FPGA for accelerating computation-intensive workloads.

Lastly, AccD compiler integrates an intelligent **design space explorer** *(DSE)* to pinpoint the "optimal" design for different problem settings. In general, there are no existing "one size fits all" solution: the best configuration for algorithmic and hardware optimization would differ across different distance-related algorithms, or even different inputs of the same distance-related algorithm. To this end, DSE utilizes a light-weighted performance and resource modeling, and the genetic algorithm to produce a high-quality optimization configuration automatically and efficiently.

Overall, our contributions are:

- We propose the first optimization framework that can automatically optimize and generate high-performance and power-efficient designs of distance-related algorithms on CPU-FPGA heterogeneous computing platform.
- We develop a Domain-specific Language, DDSL, to unify different distance-related algorithms in an effective and succinct manner, laying the foundation for general optimization across different problems.
- We build an optimizing compiler for the DDSL, which automatically reconciles the benefits from both algorithmic optimization on CPU and hardware acceleration on FPGA.
- The intensive experiments on several popular algorithms across a wide spectrum of datasets show that AccD-generated CPU-FPGA designs could achieve 24.17× speedup and 77.98× better energy-efficiency on average compared with standard CPU-based implementations.

2. Related Work

Previous research accelerates distance-related algorithms in two aspects: *Algorithmic Optimization* and *Hardware Acceleration*. More details are discussed in the following subsections.

2.1. Algorithmic Optimization

From the algorithmic standpoint, previous research highlights two optimizations: 1) KD-tree based optimization [23–27], which relies on storing points in special data structures to enable nearest neighbor search without computing distances to all target points. These methods often deliver $3\times \sim 6\times$ performance improvement [23–27] compared with the unoptimized versions in low dimensional space, while suffering from a serious performance degradation when handling large datasets with high dimension ($d \geq 10$) due to their exponential memory and computation overhead; 2) TI based optimization [14, 15, 17, 18], which aims at replacing computation-expensive distance computations with cheaper bound computations, demonstrates its flexibility and scalability. It can not only reduce the computation complexity at the different levels of granularity but is also more adaptive and robust to the datasets with a wide range of size and dimension. However, most existing works focus on one specific algorithm (e.g., KNN [18], KMeans [14, 15], and etc.), which lack of extensibility and generality across different distance-related problems.

An exception is a recent work, TOP [17], which builds a unified framework to optimize various distance-related problem with pure TI optimization on CPU. Our work shares a similar high-level motivation with their work, but targeting at a more challenging scenario—algorithmic and hardware co-optimization on CPU-FPGA platform. In this case, we can not reuse the algorithmic optimization from TOP, as the best algorithmic optimization on CPU would work even inferior on CPU-FPGA platform compared with the standard version w/o algorithmic optimization due to the severe computation irregularity and extra memory overhead.

2.2. Hardware Acceleration

From the hardware perspective, several FPGA accelerator designs have been proposed, but still suffer from some major limitations.

First, previous FPGA designs are generally built for specific

distance-related algorithm and hardware. For example, works from [10–12] targeting on KNN FPGA acceleration, while researches from [13, 19, 20] focusing on KMeans. Moreover, previous designs [10, 11] usually assumes that datasets can be fully fit into the FPGA on-chip memory, and they are only evaluated on a limited number of small datasets, for example, [11] for KMeans acceleration is evaluated on a micro-array dataset with only 2,905 points. These designs often encounter portability issue when transferring to different settings. Besides, these "hard-coded" design and optimization create the difficulties for a fair comparison among different designs, which hampers the potential work and studies in the future.

The second problem with previous works is that they fail to incorporate algorithmic optimization in the hardware design. For example, works from [10, 12, 13, 19], directly port the standard KMeans and KNN algorithms to FPGA, and only apply hardware-level optimization. One exception is a recent work [28], which combines the triangle-inequality and FPGA acceleration for KMeans. It gives a considerable speedup compared to state-of-the-art methods, showcasing the great opportunity of applying algorithm-hardware co-design and co-optimization. Nevertheless, this idea is far from explored, possibly because it requires the domain knowledge and expertise from both the algorithm and hardware to combine both of them effectively.

In addition, previous work largely focuses on the traditional hardware design flow, which requires long implementation cycle and huge manual efforts. For example, work from [10, 12, 16, 21, 22, 29–31] builds the design based on the traditional VHDL/Verilog design flow, which requires hardware expertise and over course of months arduous development. In contrast, our AccD design flow brings significant advantages of programmability and flexibility due to its high-level OpenCL-based programming model, which minimizes the user involvement in the tedious hardware design process.

3. Distance-related Algorithm Domain-Specific Language (DDSL)

Distance-related algorithms share commonalities across different application domains and scenarios, even though they look different in their high-level algorithmic description. Therefore, it is possible to generalize these distance-related algorithms. AccD framework defines a DDSL, which provides a high-level programming interface to describe distance-related algorithms in a unified manner. Unlike the API-based programming interface used in the TOP framework [17], DDSL is built on C-like language, and provides more flexibility in low-level control and performance tuning support, which is crucial for FPGA accelerator design.

Specifically, DDSL utilizes several constructs to describe the basic components (**Definition**, **Operation**, and **Control**) of the distance-related algorithms, and also identify the potential parallelism and pipeline opportunities at the design time. We detail these constructs in the following part of this section.

3.1. Data Construct

Data construct is a basic **Definition Construct**. It leverages `DSet` primitive to indicate the name of the data variable, and the `DType` primitive to notate the type characters of defined variable. Data construct serves as the basis for AccD compiler to understand the algorithm description input, such as the data points or clusters that are used in the distance-related algorithms. An example of data construct is shown in the code below, where we define the variable and dataset using DDSL data construct.

```
/* Define a single variable */
DVar [setName] DType [Optional_Initial_Value];
/* Define the matrix of dataset */
DSet [setName] DType [size] [dim];
```

In most distance-related algorithms, the dataset can be defined as the source set and the target set. For example, in KMeans, the source set is the set of data points, and the target set is the set of clusters; In Nbody simulation, the source and the target are the same set of data points. Currently, AccD support several data types including the *int* (32-bit), *float* (32-bit), *double* (64-bit) based on the users requests, algorithm performance, and accuracy trade-offs.

3.2. Distance Compute Construct

Distance computation is the core **Operation Construct** for distance-related algorithms, which measures the exact distance between two different data points. This construct requires several fields, including data dimensionality, distance metrics, and weight matrix (if weighted distance specified).

```
AccD_Comp_Dist(Input p1, Input p2, Output disMat,
    Output idMat, Dim dim, Met mtr, Weg mat)
```

p1, p2	Input data matrix. ($n_1 \times d$, $n_2 \times d$)
disMat	Output distance matrix. ($n_1 \times n_2$)
idMat	Output id matrix. ($n_1 \times n_2$)
dim	Dimensionality of input data point.
mtr	Distance metric: `(Weighted|Unweighted) (L1|L2)`
mat	Weight matrix: Only used for weighted distance ($1 \times d$)

Table 1: Distance Compute Construct Parameters.

3.3. Distance Selection Construct

Distance selection construct is an **Operation Construct** for distance value selection and it returns the Top-K smallest or largest distances and the corresponding points ID number from the provided distance and ID list. This construct helps AccD compiler to understand the distances of users interests.

```
AccD_Dist_Select(Input distMat, Input idMat,
    Output qkSet, Range ran, Scope scp)
```

srcKMat	Top-K id matrix ($n_1 \times k$)
ran	Scalar value of `K` (e.g., KMeans, KNN) or distance threshold (e.g., Nbody Simulation)
scp	Top-K `(smallest|largest)` values

Table 2: Distance Selection Construct Parameters.

3.4. Data Update Construct

Data update construct is an **Operation Construct** for updating the data points based on the results from the prior constructs. For example, KMeans updates the cluster centers by averaging the positions of the points inside. This construct requires the variable to be updated and the additional information to finish this update, such as the point-to-cluster distances. The status of this data update will be returned after the completion of all its inside operations. The status variable is to tell whether the data update makes a difference or not.

```
AccD_Update(Update var, Input p1 ,..., Input pm,
    Status s)
```

upVar	Input data/dataset to be updated
p1, ..., pm	Additional information used in update
S	Status of update operation.

Table 3: Data Update Construct Parameters.

3.5. Iteration Construct

Iteration construct is a top-level **Control Construct**. It is used to describe the distance-related algorithms which requires iteration, such as KMeans. Iteration construct requires users to provide either the maximum number of iteration or other exiting condition.

```
AccD_Iter(maxIterNum|exitCond){
   subConstruct sc1;
   subConstruct sc2;
   ...
   subConstruct scn;
}
```

3.6. Example: KMeans

To show the expressiveness of DDSL, we use KMeans as an example. From the code shown below, with no more than 20 lines of code, DDSL can capture the key components of user-defined KMeans algorithm, which is essential for AccD compiler to optimize and generate the design for CPU-FPGA platform.

```
/* Define the variables and datasets */
DVar K int 10;
DVar D int 20;
DVar psize int 1400;
DVar csize int 200;
DSet pSet float psize D;
DSet cSet float csize D;
DSet distMat float psize csize;
DSet idMat int psize csize;
DSet pkMat int psize K;

AccD_Iter(S){
   S = false;
   /* Compute the inter-dataset distances */
   AccD_Comp_Dist(pSet, cSet, distMat, idMat, D,
       "Unweighted L1", 0);
   /* Select the distances of interests */
   AccD_Dist_Select(distMat, idMat, K, "smallest",
       pkMat);
   /* Update the cluster center */
   AccD_Update(cSet, pSet, pkMat, S)
}
```

4. Algorithm Optimization

This section explains a novel triangle inequality (**TI**) optimizations tailored for CPU-FPGA platform. TI has been used for optimizing distance-related problems, but often on the sequential processing systems. Our design features an innovative way of applying TI to obtain low-overhead distance bounds for unnecessary distance computation elimination while maintaining the computation regularity to further ease hardware acceleration on FPGA.

4.1. TI in Distance-related Algorithm

As a simple but powerful mathematical concept, triangle inequality (**TI**) has been used to optimize the distance-related algorithm. Figure 2a gives an illustration. It states that $d(A,B) < d(A,L_{ref}) + d(L_{ref},B)$, where $d(A,B)$ represents the distance between point A and B in some metric (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) and upper bound (ub) of the distance between two points. This is also the standard and most common usage of TI for deriving bounds of distance.

In general, bound can be used as a substitute for the exact distance in the distance-related data analysis. Take the Nbody simulation as an example, it is to find target points that are within R (the radius) from the given query points. Suppose we get the $lb(q,t) = 10$ and $10 > R$, then we are 100% confident that point t is not within R of query point q. As a result, there is no need to compute the exact distance between point q and t. Otherwise, the exact distance computation will still be needed to be carried out for direct comparison. While many previous research [11, 14, 15, 32, 33] gains success in directly porting

the above point-based TI to optimize distance-related algorithms, they usually suffer from memory overhead and computations irregularity, which results in inferior performance.

4.2. Generalized Triangle Inequality (GTI)

AccD uses a novel Generalized TI (GTI) to remove redundant distance computation. It generalizes the traditional point-based TI while significantly reducing the overhead of bound computation. The traditional point-based TI focuses on tighter bound (more closer to exact distance) to remove more distance computation, but it induces the extra bound computations, which could become the new computation bottleneck even after many distance calculations being removed. In contrast, GTI strikes a good balance between distance computation elimination and bound computation overhead. In particular, AccD highlights GTI from three perspectives: *Two-landmark bound computation*, *Trace-based bound computation*, and *Group-level bound computation*.

Two-landmark Bound Computation Two-landmark scheme aims at reducing the bound computation through effective distance reuse. In this case, the distance bound between two points can be measured through two landmarks as the reference points. As illustrated in the Figure 2b, the distance bound between point A and B can be computed based on the existing $d(A,A_{ref})$, $d(B,B_{ref})$ and $d(A_{ref},B_{ref})$ through the following Equation 1, where A_{ref} and B_{ref} are the landmark points for point A and B, 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 query and target point set. In this case, much fewer bound computations are required compared with the one-landmark case (shown in Figure 2a). This can also be validated through a simple calculation. Assuming in KNN-join, we have m query points, n target points, z_{qry} query landmarks, and z_{trg} target landmarks. Also, we have $z_{qry} << m$ and $z_{trg} << n$ in general. Therefore, we can get $m+n+z_{qry} \times z_{trg}$ bound computations for Two-landmark case, which is much smaller than one-landmark bound computation ($m \times z_{trg} + n$ or $n \times z_{qry} + m$).

Trace-based Bound Computation Trace-based bound computation specializes its strength in iterative distance algorithms with points update, since it can largely reduce the bound computation overhead over numbers of iterations. The key of Trace-based bound computation is selecting the appropriate landmark points as references. For example, in KMeans, only the target pch iteration, therefore, we can choose the previous positions of clusters from last iteration as the landmarks for bound computation in the current iteration, since those "old" cluster positions can be close enough to current point position to offer "tight" bound. This process can be illustrated in the Figure 2c, the distance bound $d(A,B')$ can be calculated based $d(B,B')$ and $d(A,B)$, where the B' is the new point position while B is the point old position from the last iteration.

In addition, the Trace-based bound computation can also work collaboratively with the Two-landmark cases. For example, in the Nbody simulation, the source and target points are essentially the same dataset, and would get updated across iterations. We can choose the "old" positions of each point from the last iteration as the landmark for the bound computation at the current iteration, due to its closeness towards the current point. This case can be clarified in the Figure 2d, where A and B are the "old" point positions from the last iteration, A' and B' are the new source and target point. Then based on the information of $d(A,B)$, $d(A,A')$ and $d(B,B')$, the new distance bound between the A' and B' can be easily derived with using the old point A and B as the reference points. And the cost of this is also low as $O(m+n)$, since each point in the source or target set only need to maintain the shifted distance between its new position and old position from the last iteration. In contrast, only applying Two-landmark without the effective temporal reuse of the old point position will result in at least $O(m \times z_{src} + n \times z_{trg})$. In this case, the distances between each point and all the reference points have to be computed to know its new closest landmark before applying the bound computation.

Group-level Bound Computations Group-level bound computation aims at reducing the bound computation overhead while maintaining the computation regularity. Group-level Bound Computations features itself with capability to combine with the aforementioned two bound cases as the hybrid bound computation approach. In the combination with the Two-landmark case, as shown in the Figure 2f, the points in the each group (A and B) share the same landmark (A_{ref} and B_{ref}) as the reference point. Then based on the $d(A_{ref},B_{ref})$ and the $d_{max}(a,A_{ref})$ and $d_{max}(b,B_{ref})$, we can get the group-level bound based on the Equation 2, where $d_{max}(a,A_{ref})$ and $d_{max}(b,B_{ref})$ get the distance between the farthest point within each group and its group reference point.

$$\begin{aligned} lb(A,B) &> d(A_{ref},B_{ref}) - d_{max}(a,A_{ref}) - d_{max}(b,B_{ref}) \\ ub(A,B) &< d(A_{ref},B_{ref}) + d_{max}(a,A_{ref}) + d_{max}(b,B_{ref}) \end{aligned} \quad (2)$$

In the combination with the Trace-based case, it will generate a hierarchy bound as an hybrid solution, which includes point-group bound and point-point bound computation. As exemplified in the Figure 2e, each group counts on its old group center as the landmark for reference, and each point relies on its old position as the landmark for reference. Then based on the $d(A,A')$, $d(B,B')$, $d(d,d')$, and the old distance $d(c,A)$, $d(c,B)$ and $d(c,d)$, where A and B are the point group, point d is the closest point of point c in the last iteration. And we can calculate the $lb(c,G')$ and $ub(c,d')$ based on Equation

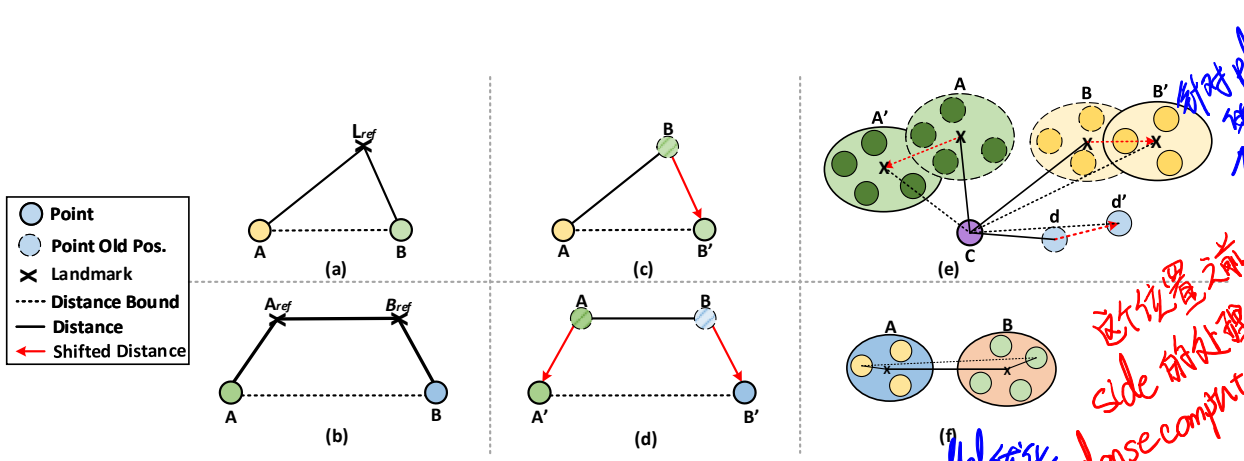


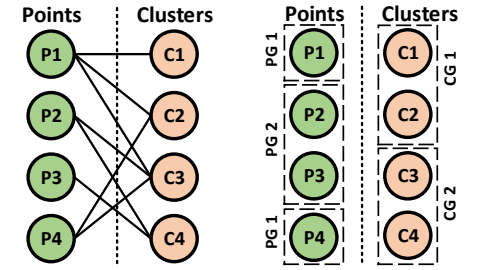
Figure 2: TI Optimization.

3,

$$lb(c,G') = d(c,G) - d_{max}(G,G') \qquad (3)$$
$$ub(c,d') = d(c,p) + d_{max}(d,d')$$

where the $G \in \{A,B\}$. If we can have the $lb(c,G') > lb(c,d')$, it is impossible that the points inside the group A' and B' can become the closest point of the c in the current iteration. Therefore, the distance computation between the point c and all points inside these groups can be safely avoided.

In addition to distance saving, group-level bound computation offers two other benefits to facilitate the underlying hardware acceleration. First, the computation regularity on the remaining distance computation will increase compared with the point-level bound computation. Since points inside each group will share the commonality in computation, which facilitates the parallelization for acceleration. For example, points inside the same source group will always maintain the same candidates of target points for distance computation after the group-level filtering, as shown in the Figure 3b, whereas the point-level bound computation usually results in the large divergence of distance computation among different points, as shown in the Figure 3a, which is a killer of parallelization and pipelining. Second, group-level bound computation brings the



Point	Candidate Clusters
P1	C1, C2, C3
P2	C3, C4
P3	C4
P4	C2, C3

PG	Candidate Groups
PG1	Group 1, Group2
PG2	Group2

Figure 3: Point-to-cluster distance computation (a) without grouping; (b) with grouping.

benefits of reducing memory overhead. Assuming we have n query points and m target points, z_{src} source group and z_{trg} target group, the memory overhead of maintaining distance bounds is $O(n \times m)$ in the traditional point-level bound computation case. However, in the group-level bound computation case, we only have to maintain distance bounds among groups, the memory overhead is $O(z_{src} \times k_c)$, where $z_{src} << n$ and $z_c << m$. Therefore, in terms of memory efficiency, group-level bound computation can outperform the point-level bound computation to a great extent.

5. Architecture Design

AccD design is built on the CPU-FPGA architecture, which highlights its significant computation performance ($1 \sim 5$ Tflops) and high bandwidth performance ($10 \sim 30$ Gbps). And it has been widely adopted as the modern data center solution for high-performance computing and acceleration. The host-side application of AccD design is responsible for data grouping and distance computation filtering, which consist of complex operations and execution dependency, but lack of pipeline and parallelism. On the other hand, the FPGA-side of AccD design is built for accelerating the distance computations, which is composed of simple and vectorizable operations.

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

5.1. Memory Optimization

After applying the TI-based filtering to remove the redundant distance computation, different data points will have different candidate points or clusters, which causes the distance computation irregularity, as shown in Table 4. In this case, the

memory access of the source points and the target (candidate) points/clusters would hurdle the overall distance computations performance, as shown in the Figure 4a. There are two reasons: 1) Irregularity in accessing the source and targeted point leads to frequent low-efficiency memory access, since the data points of interest are stored at disjoint memory space that requires multiple separated read operations to fetch them all; 2) Irregularity in computation reduce the chance of operation parallelism, since the largely diverged computation jeopardizes the benefits from parallelism. As a consequence, directly offloading the computation-intensive workload to the FPGAs is far from reaching desirable performance. Therefore, at the memory architecture level, we re-organize the source points and target points, in the consecutive memory and leverage the highly efficient matrix multiplication for efficient distance computation.

Table 4: Non-optimized Mem.

Src cg ID	Trg cg ID
1	2, 4, 6
3	8, 10, 12
...	...
5	2, 4, 6
6	8, 10, 12

Table 5: Optimized Mem.

Src cg ID	Trg cg ID
1	2, 4, 6
5	2, 4, 6
3	8, 10, 12
6	8, 10, 12
...	...

As shown in Table 5, AccD source points, which have the same set of target candidate points or clusters, are placed to the continuous memory address to maximize the memory access efficiency. Similarly, the corresponding target points are placed consecutively for efficient access. This continuous data placement strategy can combine multiple external memory access into a single burst access known as the memory coalescing, which ensures the efficient use of available external memory bandwidth with less contention for memory access between multiple computational blocks, as shown in Figure 4b. Moreover, AccD stacks the data points which have the same set of the candidate point/cluster groups into an input matrix, and points in the candidate groups as another input matrix, and use the highly-regularized matrix computation for considerable performance on FPGA accelerator.

5.2. Distance Computation Kernel

Distance computation takes the major time complexity in distance-related algorithms. In AccD, after TI filtering, the remaining distance computation is accelerated on FPGA. To leverage the OpenCL kernel for distance computation, points involved in the remaining distance computations are organized into two sets: source set and target set. AccD spots the efficient way of distance computation that can benefit the underlying hardware. It decomposes the original point-to-point distance computation in to three parts, as shown in the Equation 4.

$$(A-B)^2 = A^2 - 2*A\cdot B + B^2 \quad (4)$$

where A^2 or B^2 only takes the $O(m\times d)$ and $O(n\times d)$, while

Cg ID	Points ID
1	3, 8, 9
2	5, 6, 7
3	1, 2, 4
...	...

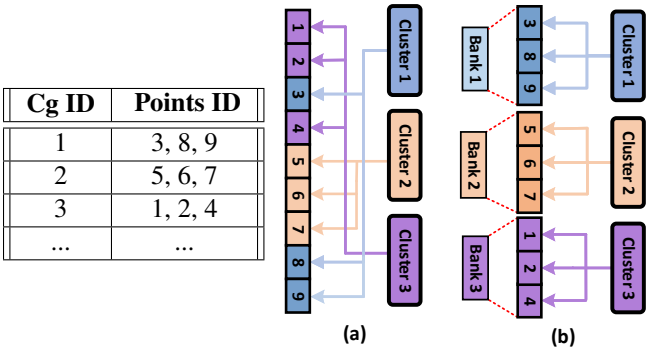


Figure 4: (a) Memory Non-aligned;(b) Memory aligned.

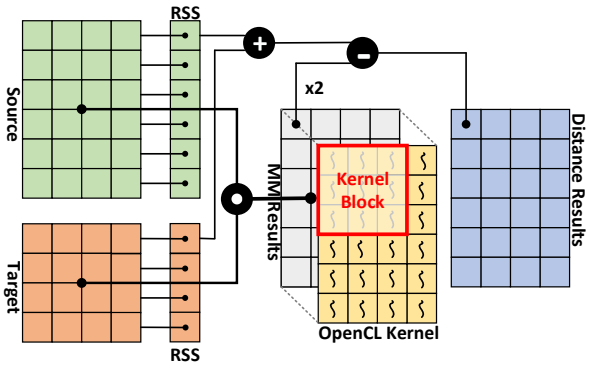


Figure 5: AccD Matrix-based Distance Computation.

the $A\times B$ takes $O(m\times n\times d^2)$, which dominates the overall computation complexity. In our AccD framework, the results of A^2 and B^2 can be pre-computed and stored for direct memory access when generating the final distance results, while the most time-consuming part $A\cdot B$ can be accelerated on FPGA. This computation process can be described as the Figure 5, the source and target point Row-wise Square Sum (RSS) is pre-computed through in a fully-parallel manner. And the vector multiplication between each source and target point is mapped to an OpenCL kernel thread for a fine-grained parallelization. Moreover, a block of threads, as highlighted in the "red" square box of Figure 5, is the kernel thread workgroup, which can share a part of the source and target point to increase the on-chip data locality. Based on this kernel organization, AccD hardware architectural design offers several tunable hyperparameters for performance and resource trade-off: the size of kernel block, the number of parallel pipeline in each kernel block, and etc. To efficiently find the "optimal" parameters that can maximize overall performance while respecting the constraints, we harness the AccD explorer for efficient design space search, which is detailed in Section 6.2.

6. AccD Compiler

In this section, we detail our AccD compiler in two aspects: design parameter and constraints, and design space exploration.

6.1. Design Parameters and Constraints

AccD uses a parameterized design strategy for better design flexibility and efficiency. It takes the design parameters and constraints from software and hardware to explore and locate the "optimal" design point tailored for specific application scenarios. At the software level, the number of groups affects distance computation filtering performance. At the hardware level, there are five parameters: 1) Size of computation block, which decides the size of data shared by a group of computing elements; 2) SIMD factor, which decides the number of computing elements inside each computation block; 3) Unroll factor, which tells the degree of parallelization in each single distance computation; 4) Clock frequency, which decides the operation speed of FPGA; 5) Memory bandwidth, which tells the speed of data communication between host memory and FPGA on-chip memory. In addition, there are several hardware constraints, such as on-chip memory size, the number of logic units, and the number of registers. All of these parameters and constraints are included in AccD analytical model for design exploration, which will be discussed in the next subsection.

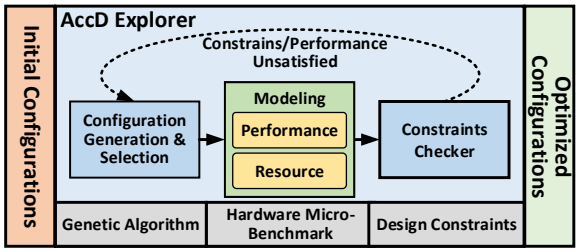


Figure 6: AccD Explorer

6.2. Design Space Exploration

Finding the best combination of the design parameters under the given hardware constraints demands non-trivial efforts in the efficient design space exploration. AccD leverages an optimization-function based strategy combined with genetic algorithm (GA) to facilitate the design space search. The goal of AccD explorer is to deliver a performance-resource balanced accelerator design that can satisfy the users' requirements.

Performance Modeling The first step of AccD explorer is to model the design performance in terms of design latency from AccD compiler algorithm analysis results. We formulate the latency function of AccD design as the Equation 5,

$$Latency = Latency_{filt} + Latency_{comp} \tag{5}$$

where the $Latency_{filt}$ and $Latency_{comp}$ are the time of group-based filtering process and remaining distance computation, respectively. And they can be calculated as the Equation 6,

$$\begin{aligned} Latency_{filt} &= \frac{n_{cg} \times src_{size} \times trg_{size} \times d \times k}{n_{iteration}} \\ Latency_{comp} &= \frac{src_{size} \times trg_{size} \times ratio_{save} \times d}{blk^2 \times frequency \times unroll \times simd} \end{aligned} \tag{6}$$

where the n_{cg} is the number of cluster groups; src_{size} and trg_{size} are the number of points inside source and target set, respectively; d is the data dimensionality; $n_{iteration}$ is the number of grouping iteration. $ratio_{save}$ is the saved distance after group-based filtering; blk is the computation block size; $frequency$ is the FPGA design clock frequency; $unroll$ is the computation unroll factor of single point-to-point distance; $simd$ is the number of parallelized worker inside each computation block.

As another important factor in design modeling, the required bandwidth BW can be calculated as the Equation 7,

$$BW = \frac{(src_{size} + trg_{size}) \times d \times size_{data\_type}}{Latency} \tag{7}$$

where the $size_{data\_type}$ can be either 32-bit for *int* and *float* or 64-bit for *double*. The saving ratio ($ratio_{save}$) measures the group-based filtering efficiency, as illustrated in the Equation 8.

$$ratio_{save} = \textbf{Filtering}(src_{size}, trg_{size}, d, n_{cg}, n_{iteration}) \tag{8}$$

where $ratio_{save}$ is calculated by a *Filtering* function, which combines the algorithm profiling results and the current dataset setting.

Constraints of the accelerator design can be formulated as the Equation 9, which includes the *Mem* (the size of FPGA on-chip memory), *BW* (the bandwith of data communication between external memory and on-chip memory), *Computing_Unit* (the number of computing units) and *Logic_Unit* (the number of logic units):

$$\begin{aligned} BW &\leq BW_{max}, \\ Mem &\leq Mem_{max}, \\ Computing\_Unit &\leq Computing\_Unit_{max}, \\ Logic\_Unit &\leq Logic\_Unit_{max} \end{aligned} \tag{9}$$

Resource Modeling Directly measuring the hardware resource usage of the accelerator design from high-level algorithm description is challenging because of the hidden transformation and optimization in hardware design suite. However, AccD uses a micro-benchmark based methodology to measure the hardware resource usage by analytical modeling. The major hardware resource consumption of AccD comes from the distance computation kernel, which depends on the design factors including the kernel block size, the number of SIMD workers, unroll factor, and the source and target point size and dimension.

AccD builds the analytical model through micro-benchmark. In AccD resource analytical model, the design factors are classified into two categories: dataset-independent and dataset

independent factors. The main idea behind the AccD strategy is to get the exact hardware resource consumption statistics through micro-benchmark on the hardware designs with different dataset-independent factors. For example, we can benchmark a single distance computation kernel with different combination values of block size and number of SIMD workers within each block to get its statistics. Since these two factors are dataset-independent, which can be decided before knowing the dataset details. However, to estimate the resource consumption for datasets with different size and dimensionality, AccD leverages the resource-estimation formula based on the kernel organization and the miro-benchmark statistics of each single kernel, during the runtime programming of the FPGA, host program will decide the number of kernels and its organization in multi-dimension space based on the input dataset size and dimension. Therefore, the overall hardware resource consumption can be calculated as the Equation 10.

$$Res\_est = Res_{single} \times \textbf{ceil}(\frac{src_{size}}{blk}) \times \textbf{ceil}(\frac{trg_{size}}{blk}) \quad (10)$$
$$Res \in \{Mem, Computing\_Unit, Logic\_Unit\}$$

Based on the estimated results for each type of resource, AccD can check the maximum number of each type of resource to decide whether keep or discard the current design.

Design Optimization The optimization of an accelerator design, as shown in Equation 11 includes the maximize the distance computation saving by using triangle-inequality based filtering while minimizing the latency of the overall design.

$$Opt. = \{Max(ratio_{save}), Min(Latency), Min(BW)\}$$

AccD applies a simple genetic algorithm (GA) iteratively to get the best design configuration (parameter combination). There are three steps in each iteration: First, AccD explorer generate a set of design configurations; Second, AccD explorer applies analytical modeling by using the Equation 5, 7, and 8 on these generated configurations to estimate their performance and resource; Third, AccD explorer only keeps the configurations that can potentially maximize the computation saving ratio while minimizing the overall design latency and the required memory bandwidth. Forth, AccD explorer will "crossover" these kept design configurations to generate a new set of design configuration for the next iteration. This design exploration process will stop until the performance difference of generated configurations from two consecutive iterations is smaller than a given threshold value, thus, we can get an optimal or near-optimal design configuration under the current constraints.

7. Evaluation

In this section, we choose three representative benchmarks (KMeans, KNN-join, Nbody Simulation) and evaluate their corresponding AccD design on the CPU-FPGA platform.

KMeans KMeans [7, 24, 34–36] 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 then update the clusters based on the average position of their inside points. We choose it as our benchmark since it can show the benefits of AccD hierarchy (**Trace-based + Group-level**) bound computation optimization on iterative algorithm with disjoint source and target set.

KNN-join KNN-join Search [8, 25, 37] finds the Top-K nearest neighbor points for each point in source set from target set. It first computes the distance between each source point and the all target points. Then it ranks the K-smallest distances for each source point and get its corresponding closest Top-K target points. KNN-join can help to demonstrate the effectiveness of AccD hybrid (**Two-landmark + Group-level**) bound computation optimization on non-iterative algorithm.

Nbody Simulation Nbody Simulation [38, 39] mimics the particle movement within a certain range of 3D space. In each time step, it first computes the distances between each particles and all of its neighbors, and then calculates acceleration and new position of the particles based on these distances. While Nbody simulation is also iterative, it has several differences compared with KMeans algorithm: 1) Nbody simulation has the same dataset (particles) for source and target set, whereas KMeans operates on different source (point) and target (cluster) set; 2) All points in the Nbody simulation would change their positions according to the time variation, whereas in KMeans only the target set (cluster) would change their positions during the center update; 3) Nbody simulation has the same size of source and target set, whereas KMeans target set (cluster) is much smaller than source set (point) in general. Nbody simulation can help us to show the strength of AccD hybrid bound computation (**Two-landmark + Trace-based + Group-level**) on iterative algorithm with the same source and target set.

7.1. Experiment Setup

Tools and Metrics In our evaluation, we use the Intel Stratix 10 DE10-Pro [40] as the FPGA accelerator and run the host side software program on Intel Xeon Silver 4110 processor [41] (8-core 16-thread, 2.1GHz base clock frequency, 85W TDP). The 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 implement AccD design on DE10-Pro by using Intel Quartus Prime Software Suite [42] with Intel OpenCL SDK included. To measure the system power consumption (Watt) accurately, we use the off-the-shelf Poniie PN2000 as the external power meter to get the runtime power of the Xeon CPU and the DE-10 Pro FPGA, and calculate the dynamic power using the following Equation 12,

$$\textbf{Power}_{\textbf{dynamic}} = Power_{runtime} - Power_{static} \quad (12)$$

KMeans				KNN-join			Nbody Simulation	
Dataset	**Size**	**Dimension**	**#Cluster**	**Dataset**	**Dimension**	**#Source**	**Dataset**	**#Particle**
Poker Hand	25,010	11	158	Harddrive1	64	68,411	P-1	16,384
Smartwatch Sens	58,371	12	242	Kegg Net Directed	24	53,413	P-2	32,768
Healthy Older People	75,128	9	274	3D Spatial Network	3	434,874	P-3	59,049
KDD Cup 2004	285,409	74	534	KDD Cup 1998	56	95,413	P-4	78,125
Kegg Net Undirected	65,554	28	256	Skin NonSkin	4	245,057	P-5	177,147
Ipums	70,187	60	265	Protein	11	26,611	P-6	262,144

Table 6: Datasets for Evaluation.

where the static power is measured when the CPU-FPGA platform is running without any workload, while the runtime power is measured during the execution of the distance-related algorithm. We get the runtime power by repetitively running the same task (the same distance related algorithm on the same dataset), and read the power value until it is stable. And we calculate the energy efficiency by using the Equation 13,

$$\textbf{Energy\_Efficiency} = \frac{Time_{other}/Time_{baseline}}{Power_{baseline}/Power_{other}} \quad (13)$$

where the $Time_{other}$ and $Power_{other}$ stand for the execution time and power consumption of the optimized implementation, such as TOP and AccD.

Table 7: Implementation Description.

Name	Techniques	Description
Baseline	Standard Algorithm without any optimization, CPU.	Naive for-loop based implementation on CPU.
TOP	Point-based Triangle-inequality Optimized Algorithms, CPU.	TOP [17] optimized distance-related algorithm running on CPU.
CBLAS	CBLAS library Accelerated Algorithms, CPU.	Standard distance-related algorithm with CBLAS [43] acceleration.
AccD	Algorithmic-hardware co-design, CPU-FPGA platform.	GTI filtering and FPGA acceleration of distance computations.

Implementations The CPU-based implementations consist of three types of programs: the naive for-loop sequential implementation without any optimization (selected as our **Baseline** to normalize the speedup and energy-efficiency), the algorithm optimized by **TOP** [17] framework and the algorithm optimized by **CBLAS** [43] computing library. Note that the TOP + CBLAS implementation is **not** included in our evaluation, since after applying TOP point-based TI filtering, each point in the source set has a distinctive list of points from the target set for distance computation, whereas CBLAS requires such uniformity in the distance computations. Therefore, it is challenging to combine TOP optimization and CBLAS acceleration together.

Dataset In the evaluation, we use six datasets for each algorithm. The selected datasets can cover the wide spectrum of mainstream datasets, including the datasets from UCI Machine Learning Repository [44], and the datasets have ever been used by the previous paper [15, 17, 18] in the related domains. Details of these datasets are listed in the Table 6. Please note KNN-join algorithm will find the Top-1000 closest neighbor of each query point.

7.2. Comparison with Software Implementation

Performance Comparison As shown in the Figure 7, TOP, CBLAS and AccD achieve average 9.12×, 9.18× and 31.42× compared with Baseline implementation across all algorithms and dataset settings, respectively. As we can see, AccD design can always maintain the highest speedup among these implementations. This is largely dues to AccD GTI optimization in reducing distance computation and its efficient hardware acceleration of the distance computation on FPGA.

We also observe that TOP implementation shows its strength for large datasets. For example, on dataset 3D Spatial Network ($n = 434,874$) in KNN-join, TOP implementation achieves 39.78× speedup. Since the fine-grained point-based TI optimization of TOP can reduce most (more than 90%) of the unnecessary distance computations, which benefits the overall performance to a great extent. Note that the intrinsic point distribution of the dataset would also affect the filtering performance of TOP, but in general, the larger dataset could lead TOP to spot and remove more redundant computations.

What we also notice is that CBLAS implementation demonstrates its performance on datasets with relatively high dimensionality. For example, on dataset KDD Cup 2004 ($d = 74$) in the KMeans algorithm, CBLAS achieve 11.78× speedup over the CPU Baseline, which is higher than its performance on other KMeans datasets. This is because, on high dimension dataset, CBLAS implementation can get more benefits of parallelized computing and more regularized memory access, whereas, in low dimension settings, the same optimization can only yield minor speedup.

Our AccD design achieves considerable speedup on datasets with large size and high dimensionality. For example, on dataset KDD Cup 2004 ($n = 285,409, d = 74$) and Ipums ($n = 70,187, d = 60$) in KMeans, AccD achieves 51.61× and 66.61× speedup over the baseline, and also significantly higher than both TOP and CBLAS implementations. This con-

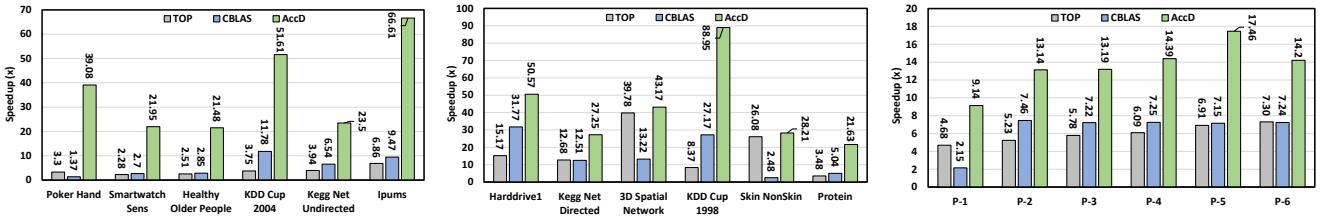


Figure 7: Performance Comparison (TOP, CBLAS, AccD): (a) KMeans (b) KNN-Join (c) Nbody Simulation. Note: Speedup is normalized w.r.t Baseline.

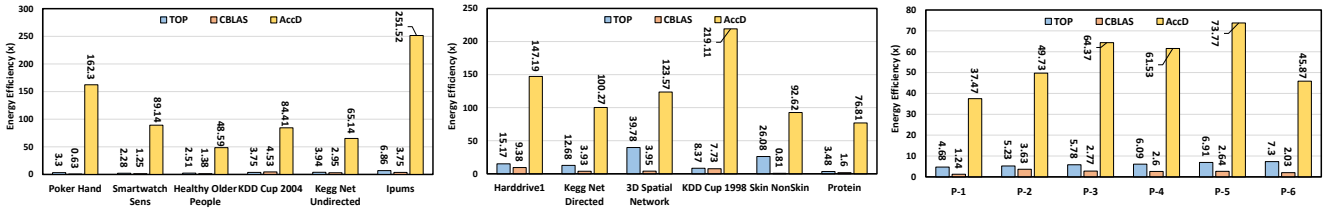


Figure 8: Energy Efficiency Comparison (TOP, CBLAS, AccD): (a) KMeans. (b) KNN-Join. (c) Nbody Simulation. Note: Energy Efficiency is normalized w.r.t Baseline.

clusion can also be extended to KNN-join, such as 88.95× speedup on dataset KDD Cup 1998 ($n = 95,413$, $d = 56$). Since our AccD design can effectively reconcile the benefits from both the GTI optimization and the FPGA acceleration, where the former provides the opportunity to reduce the distance computation at the algorithm level, and the latter boosts the performance from hardware acceleration perspective. More importantly, our AccD design can balance the aforementioned two benefits to maximize the overall design performance.

Energy Comparison The energy efficiency of AccD design is significant. For example, on the KMeans algorithm, AccD designs deliver an average 116.85× better energy efficiency compared with the Baseline implementation, which is significantly higher than TOP and CBLAS implementations. There are namely two reasons behind these results: 1) Much lower power consumption. AccD CPU-FPGA design only consumes $5w \sim 17.12w$ across all algorithm and dataset settings, whereas Intel Xeon CPU consumes at least $20.9w$ and $42.49w$ on TOP and CBLAS implementation, respectively; 2) Considerable performance. AccD design achieves much better speedup (more than 5× on average) compared with the TOP and CBLAS implementation, which also contributes to overall design energy-efficiency.

Among these implementations, CLBAS 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 the much higher power consumption (average $65.79w$). TOP only leverages the single-core processing capability of the CPU and achieve moderate performance with effective distance computation reduction, which results in less power consumption (average $25.59w$) and higher energy efficiency (average 5.12×) compared with Baseline. Different from the TOP and CBLAS implementation, AccD design is built upon a low-power platform with considerable performance, which shows a far better energy-performance trade-off.

7.3. Performance Benefits Analysis

To analysis the performance benefits of AccD CPU-FPGA design in detail, we use the KMeans as the example algorithm for study. Specifically, we build four implementations for comparison: 1) TOP KMeans on CPU; 2) TOP KMeans on CPU-FPGA platform; 3) AccD KMeans on CPU; 4) AccD KMeans on CPU-FPGA platform. Note that TOP KMeans is designed for sequential-based CPU, and no publicly available TOP implementation on CPU-FPGA platform. For a fair comparison, we implement the TOP on CPU-FPGA platform with optimizations, such as memory coalescing of a list of target points, since it can improve data reuse and memory access performance, while facilitating a more efficient vector-matrix based distance computation on FPGA after the point-based TI filtering on CPU.

And we compute the normalized speedup performance of each implementation w.r.t the naive for-loop based KMeans implementation on CPU.

As shown in the Figure 9, AccD KMeans on CPU-FPGA platform can always deliver the best overall speedup performance among these four implementations. We also observe that TOP KMeans can achieve average 3.77× speedup on CPU, however, directly porting this optimization towards CPU-FPGA platform could even lead to inferior performance (av-

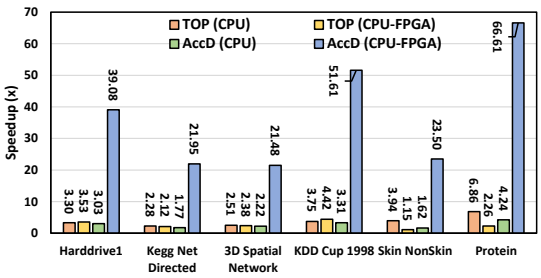


Figure 9: AccD Performance Benefits Breakdown.

erage 2.63×). Since applying the fine-grained point-based TI optimization from TOP would cause large divergence of computation for each point (i.e., each point maintain an almost "unique" list of points for distance computation after filtering).

We also notice is that AccD design on CPU achieves lower speedup (average 2.69×) compared with the TOP (average 3.77×), since its coarse-grained GTI optimization spots a fewer number of unnecessary distance computations. However, when combining AccD design with CPU-FPGA platform, the benefits of AccD GTI optimization becomes prominent (average 37.37×), since it can maintain computation regularity while reducing memory overhead to facilitate the hardware acceleration on FPGA. Whereas applying optimization to maximize the algorithm-level benefits while ignoring hardware-level properties would result in poor performance, such as the TOP (CPU-FPGA) implementation. Moreover, the comparison of AccD (CPU) and AccD (CPU-FPGA) can also demonstrates effectiveness of using FPGA as the hardware accelerator to boost the performance of the algorithms, which can deliver additional 9.68× $\sim$ 15.71× speedup compared with the software only solution.

8. Conclusion

In this paper, we present our AccD compiler framework to accelerate the distance-related algorithms on the CPU-FPGA platform. Specifically, AccD leverages a simple but expressive language construct (DDSL) to unify the distance-related algorithms, and an optimizing compiler to improve the design performance improvement from algorithmic and hardware perspective systematically and automatically. The rigorous experiments on three popular algorithms (KMeans, KNN-join, and Nbody simulation) demonstrate the AccD as a powerful and comprehensive framework for distance-related algorithms hardware acceleration on the modern CPU-FPGA platform.

References

[1] Y. Pu, J. Peng, L. Huang, and J. Chen. An efficient knn algorithm implemented on fpga based heterogeneous computing system using opencl. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2015.

[2] Hai Peng, Letian Huang, and John Chen. An efficient fpga implementation for odd-even sort based knn algorithm using opencl. In *2016 International SoC Design Conference (ISOCC)*, 2016.

[3] Yuliang Pu, Jun Peng, Letian Huang, and John Chen. An efficient knn algorithm implemented on fpga based heterogeneous computing system using opencl. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2015.

[4] Chao Wang, Lei Gong, Qi Yu, Xi Li, Yuan Xie, and Xuehai Zhou. Dlau: A scalable deep learning accelerator unit on fpga. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[5] Q. Xiao, Y. Liang, L. Lu, and S. Yan and. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on fpgas. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017.

[6] Wenlai Zhao, Haohuan Fu, Wayne Luk, Teng Yu, Shaojun Wang, Bo Feng, Yuchun Ma, and Guangwen Yang. F-cnn: An fpga-based framework for training convolutional neural networks. In *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2016.

[7] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 1982.

[8] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 1992.

[9] M. Trenti and P. Hut. N-body simulations (gravitational). *Scholarpedia*, 2008.

[10] 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.

[11] 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.

[12] 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)*, 2006.

[13] 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.

[14] Charles Elkan. Using the triangle inequality to accelerate k-means. In *Proceedings of the 20th International Conference on Machine Learning (ICML)*, 2003.

[15] 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.

[16] 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.

[17] Yufei Ding, Xipeng Shen, Madanlal Musuvathi, and Todd Mytkowicz. Top: A framework for enabling algorithmic optimizations for distance-related problems. *Proc. VLDB Endow.*, 2015.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] Dominique Lavenier. Fpga implementation of the k-means clustering algorithm for hyperspectral images. *Los Alamos National Laboratory LAUR*, 2000.

[23] 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.

[24] 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.

[25] C. Yang, X. Yu, and Y. Liu. Towards efficient knn joins on data streams. In *2014 IEEE International Congress on Big Data*, 2014.

[26] P. Sirait and A. M. Arymurthy. Cluster centres determination based on kd tree in k-means clustering for area change detection. In *2010 International Conference on Distributed Frameworks for Multimedia Applications*, 2010.

[27] Ruicheng Zhong, Guoliang Li, Kian-Lee Tan, and Lizhu Zhou. G-tree: An efficient index for knn search on road networks. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management (CIKM)*, 2013.

[28] Y. Wang, Z. Zeng, B. Feng, L. Deng, and Y. Ding. Kpynq: A work-efficient triangle-inequality based k-means on fpga. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019.

[29] Xing Zhang Zhiqiang Yang Jipan Huang Miren Tian, Xin'an Wang 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)*, 2016.

[30] 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.

[31] 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.

[32] Greg Hamerly. Making k-means even faster. In *Proceedings of the 2010 SIAM international conference on data mining (SIAM)*, 2010.

[33] H. Hong, G. Juan, and W. Ben. An improved knn algorithm based on adaptive cluster distance bounding for high dimensional indexing. In *2012 Third Global Congress on Intelligent Systems*, 2012.

[34] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 2010.

[35] Adam Coates and Andrew Y Ng. Learning feature representations with k-means. In *Neural networks: Tricks of the trade*. 2012.

[36] 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.

[37] 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.

[38] Lars Nylons. Fast n-body simulation with cuda. 2007.

[39] Shigeru Ida and Junichiro Makino. N-body simulation of gravitational interaction between planetesimals and a protoplanet: I. velocity distribution of planetesimals. 1992.

[40] Terasic de10-pro stratix 10 gx/sx fpga development kit. URL `http://www.terasic.com.tw/cgi-bin/page/archive.pl?CategoryNo=248&No=1144`.

[41] Intel xeon silver 4110 processor. URL `https://www.intel.com/content/www/us/en/products/processors/xeon/scalable/silver-processors/silver-4110.html`.

[42] Intel quartus prime software suite. `https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html`.

[43] 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.

[44] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.