PRADA: Point Cloud Recognition Acceleration via Dynamic Approximation

Zhuoran Song[1], Heng Lu[1], Gang Li[1], Li Jiang[1], Naifeng Jing[1], Xiaoyao Liang[1]
[1]*School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China*

***Abstract*—Recent point cloud recognition (PCR) tasks tend to utilize deep neural network (DNN) for better accuracy. Still, the computational intensity of DNN makes them far from real-time processing, given the fast-increasing number of points that need to be processed. Because the point cloud represents 3D-shaped discrete objects in the physical world using a mass of points, the points tend for an uneven distribution in the view space that exposes strong clustering possibility and local pairs' similarities. Based on this observation, this paper proposes PRADA, an algorithm-architecture co-design that can accelerate PCR while reserving its accuracy. We propose dynamic approximation, which can approximate and eliminate the similar local pairs' computations and recover their results by copying key local pairs' features for PCR speedup without losing accuracy. For accuracy good, we further propose an advanced re-clustering technique to maximize the similarity between local pairs. For performance good, we then propose a PRADA architecture that can be built on any conventional DNN accelerator to dynamically approximate the similarity and skip the redundant DNN computation with memory accesses at the same time. Our experiments on a wide variety of datasets show that PRADA averagely achieves $4.2\times$, $4.9\times$, $7.1\times$, and $12.2\times$ speedup over Mesorasi, V100 GPU, 1080TI GPU, and Xeon CPU with negligible accuracy loss.**

I. Introduction

Point cloud recognition (PCR) plays a key role in a wide range of applications, such as autonomous driving and robotics. An input point cloud is a collection of points that can represent 3D-shaped discrete objects with points covering the object's surface, as shown in Fig. 1(a). To recognize the points, the classic PCR algorithm works as in Fig. 1(b), and it contains three steps: 1) farthest point sampling (FPS), 2) neighbor search, and 3) feature computation. First, FPS picks the most representative points—centroid points, i.e., P_1. Next, the neighbor search operation selects multiple neighbor points for each centroid point, like P_3, and then concatenates with centroid point to form in local pairs, i.e., (P_1, P_3), (P_1, P_6). Finally, the feature computation applies a DNN on the local pairs to obtain the feature results. Note that forming the local pairs is an important step to PCR accuracy because it can recognize fine-grained patterns in complex scenes. As a result, PointNet++ [10] can achieve remarkable accuracy improvement compared to its predecessor PointNet [9], which does not focus on local pairs.

However, given a large amount of local pairs, the per local pair DNN processing in PointNet++ makes it hard for real-time PCR. Aiming for faster speed, Mesorasi [5] tries to reduce the workload by not fully processing every local pair. As illustrated in Fig. 1(b), it applies the feature computation on each input point individually, such as P_1, P_2, rather than on each local pair, i.e., (P_1, P_3), which brings significant performance improvement but at the cost of notable accuracy loss up to 7.8%.

The point cloud oftentimes represents discrete objects, so the points tend for an uneven distribution in the view space that manifests a strong clustering possibility. So, one neighbor point is likely to be accessed more than once in different local pairs, which inherently causes redundant computations in the following DNN. Based on this observation, we present PRADA, a hardware-software co-design framework. PRADA avoids the drawbacks of Mesorasi and PointNet++ by reserving the important local pairs while reducing the redundant computations.

To enable PRADA, we offer the dynamic approximation method that first detects the similar local pairs containing the same neighbor point and then approximates them equivalent. The dynamic approximation only needs to compute one of them to save redundant DNN computations. We further propose an advanced re-clustering technique that re-clusters the centroid points to constrain the approximation space. Overall, approximating the local pairs in the same cluster will speed up the DNN computation without hurting the PCR accuracy.

The second question is how to dynamically detect and approximate the local pairs considering they are produced online. We further design an end-to-end PRADA architecture. It equips a lightweight approximation engine to identify the redundancy and reconstruct the feature result. The approximation engine contains a joint dual buffer structure, which can simultaneously eliminate redundant computations and memory accesses.

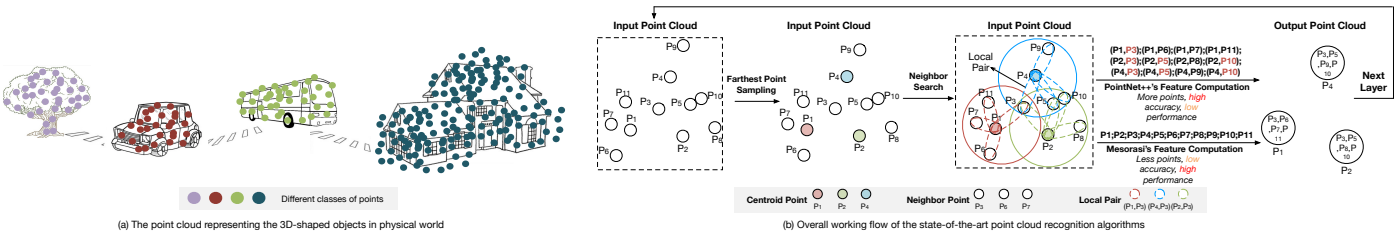
Different from the conventional PCR algorithms [9], [10], [5], PRADA **keeps** the important local pairs that can reserve the DNN accuracy, and **dynamically approximates** the similar local pairs that can accelerate the DNN processing. In contrast, conventional schemes such as Mesorasi only support **static input-point-wise** approximation which hurts accuracy. Besides, PRADA is **hardware friendly** and can support diverse point-based PCR algorithms by the end-to-end design. Moreover, PRADA offers an efficient accelerator design with a tightly coupled algorithm for PCR workloads.

II. Background and Motivation

A. NN-based PCR Algorithm

The PCR algorithm consists of multiple layers, and each layer repeatedly transforms an input point cloud to an output point cloud by FPS, neighbor search, and feature computation operations. The most time-consuming feature computation

This work is partly supported by the National Natural Science Foundation of China (Grant No. 62202288, 92264108, 61972242, 61834006). Naifeng Jing is the corresponding author.



(a) The point cloud representing the 3D-shaped objects in physical world

(b) Overall working flow of the state-of-the-art point cloud recognition algorithms

Fig. 1: An overview of point cloud recognition.

operation contains four steps—aggregation, concat, DNN, and reduction using notations listed in Table I.

The aggregation and concat operations are responsible for generating the local pairs in the form of input vectors. Specifically, the aggregation calculates the coordinate difference between the neighbor point P_i and its centroid point P_j by calculating $\mathbf{C}_i$-$\mathbf{C}_j$. Then the input feature $\mathbf{F}_i$ of P_i is concatenated with $\mathbf{C}_i$-$\mathbf{C}_j$ to form the input vector $\mathbf{I}_{i,j} = \{C_i - C_j, F_i\}$. The input vectors construct the local pair matrix (LPMatrix) and wait for subsequent DNN, which typically applies a multi-layer perceptron (MLP). Next, the MLP is executed on the input vectors and acquires the feature results to form an output feature matrix (OFMatrix). During the feature computation, the input vectors in LPMatrix share the same MLP. In the end, a reduction operation reduces the OFMatrix to a new output matrix, which becomes the output point cloud.

TABLE I: Notations.

$\mathbf{F}_i$	the input feature of point P_i
$\mathbf{C}_i$	the coordinate of point P_i
$\mathbf{C}_{i,j}$	the average coordinate of P_i and P_j
$\mathbf{I}_{i,j}$	the input vector containing $\mathbf{C}_i$-$\mathbf{C}_j$ and $\mathbf{F}_i$
$M()$	the feature result

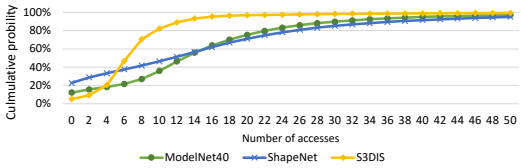
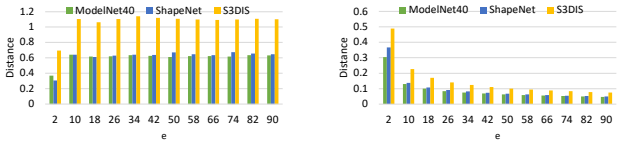


Fig. 2: Access frequency analysis.



(a) Inter-cluster distance. (b) Intra-cluster distance.

Fig. 3: Cluster distance analysis.

B. Motivation of PRADA

As depicted in Fig. 1(a), the points gather in clusters, which leads to repeated accesses to the neighbor points in the local pairs. In Fig. 1(b), P_3 appears three times in local pairs (P_1, P_3), (P_2, P_3), and (P_4, P_3). Fig. 2 quantifies the cumulative probability for a number of accesses of neighbor points in three datasets—ModelNet40, ShapeNet, and S3DIS [14], [2], [1], where 82%, 66% and 80% neighbor points have been searched more than four times, respectively. Recalling the aggregation and concat operations, such repeated accesses in neighbor points will appear as the same input feature in input vectors, creating massive similar input vectors in LPMatrix. In other words, we can approximate the similar input vectors to skip their corresponding MLP computations.

To reduce the accuracy loss by such approximation, we can exploit the spatial distribution of the centroid points by re-clustering them into e clusters and only approximate the input vectors in a limited approximation space. It works because the PCR accuracy is less sensitive to the positions of the centroid points. We then report the inter- and intra-cluster distance in Fig. 3, where the x-axis is the number of clusters. From Fig. 3(a), the inter-cluster distance is quite large. Instead, in Fig. 3(b), the intra-cluster distance is small, sharply decreasing with the increasing number of clusters. This implies that centroid points gather in clusters and can serve as a hint to partition the input vectors. With a proper number of clusters, we can reduce the massive redundancy in input vectors within each cluster while retaining accuracy.

III. PRADA Algorithm

The central idea of PRADA algorithm is to approximate the similar input vectors in LPMatrix dynamically. PRADA introduces **back-end dynamic approximation** that can eliminate the expensive feature computation by easily reconstructing their feature results. To avoid hurting the PCR accuracy, PRADA provides the **front-end advanced re-clustering** to re-cluster the centroid points.

A. The Back-end Dynamic Approximation

Dynamic Approximation. Given an input vector in LPMatrix contains the coordinate difference $\mathbf{C}_t - \mathbf{C}_i$ and input feature $\mathbf{F}_t$, where $\mathbf{F}_t$ often accounts for a larger portion (up to a 512-dimension vector in PointNet++ [10]) than the coordinate difference $\mathbf{C}_t - \mathbf{C}_i$ (a 3-dimension tuple). We can see that the input vectors containing the same feature are highly likely to be similar. We approximate them to make them equal once we dynamically detect the same feature.

Supposing point P_t is a neighbor of the centroid points P_i and P_j, their feature results after the MLP computation should be M($\mathbf{I}_{t,i}$) = M($\{\mathbf{C}_t$-$\mathbf{C}_i, \mathbf{F}_t\}$) and M($\mathbf{I}_{t,j}$) = M($\{\mathbf{C}_t$-$\mathbf{C}_j, \mathbf{F}_t\}$) . If we detect the features in two input vectors are both $\mathbf{F}_t$, we approximate them as M($\{\mathbf{C}_t$-$\mathbf{C}_{i,j}, \mathbf{F}_t\}$), where $\mathbf{C}_{i,j}$ is the average coordinate of P_i and P_j if they are adjacent in 3D space with $\mathbf{C}_i$-$\mathbf{C}_j \to 0$. As a result, the feature computations of $M(\mathbf{I}_{t,j})$ can be eliminated. In other words, the approximation makes $\mathbf{I}_{t,i}$ and $\mathbf{I}_{t,j}$ the same and only compute M($\{\mathbf{C}_t$-$\mathbf{C}_{i,j}, \mathbf{F}_t\}$) **once for acceleration**. We regard $\mathbf{I}_{t,i}$ as the reference input vector for $\mathbf{I}_{t,j}$ for later feature reconstruction.

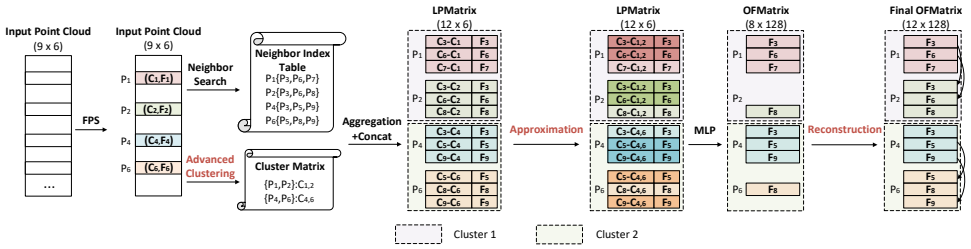


Fig. 4: PCR workflow with PRADA algorithm.

Feature Reconstruction. After the dynamic approximation, the MLP only operates on a small subset of the input vectors rather than the whole LPMatrix. To reconstruct the final OFMatrix, we will first locate the reference input vector and then retrieve its feature result to fill the OFMatrix. For example, we copy M($\mathbf{I}_{t,i}$) to the slot of M($\mathbf{I}_{t,j}$) in OFMatrix by referencing $\mathbf{I}_{t,i}$ for $\mathbf{I}_{t,j}$.

B. The Front-end Advanced Re-clustering

The dynamic approximation has minimal impact on accuracy only on the condition that the two centroid points are close to each other. For example, as in Fig. 1(b), centroid points P_1 and P_4 both have a neighbor P_3. But $\mathbf{I}_{3,1}$ and $\mathbf{I}_{3,4}$ are far from identical because P_1 is far from P_4, so the accuracy may significantly degrade if we approximate $\mathbf{I}_{3,1}$ and $\mathbf{I}_{3,4}$. Therefore, we propose to re-cluster the centroid points to constrain the approximation space.

During the FPS operation, the earlier selected centroid points are more representative than the later points [10]. Hence, we propose that in FPS, the order of how centroid points are selected can guide the centroid point re-clustering. Specifically, we first use the earlier selected e centroid points as the centers of e clusters. Next, we only have to calculate the distances between the unselected centroid points and e cluster centers and then throw the unselected points to e centers one by one. Fortunately, the neighbor search operation has already calculated the distances between all points and centroid points. As a result, we can directly leverage the distances calculated in the neighbor search operation to guide the re-clustering process. Afterwards, to ensure the difference between the approximated input vectors in a cluster as little as possible, we calculate the average coordinate of each cluster to replace the original coordinates of the centroid points in the cluster.

In conclusion, the proposed re-clustering technique reuses the results produced during the FPS and neighbor search to re-cluster centroid points without extra computing overhead.

C. PCR Example with PRADA Algorithm

Fig. 4 uses one layer to show how PCR works when inserting the proposed PRADA algorithm. This layer has 9 input points. After executing the FPS operation, 4 representative centroid points are selected. Next, each centroid point undergoes the neighbor search, and each gets 3 neighbor points. Then the feature computation operation generates the final OFMatrix.

During the feature computation, aggregation and concat operation first generates the LPMatrix by calculating the coordinate differences such as $\mathbf{C}_3$-$\mathbf{C}_1$ and concatenating feature $\mathbf{F}_3$. Next,

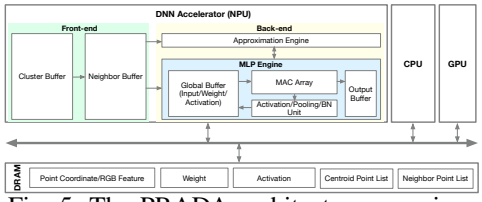


Fig. 5: The PRADA architecture overview.

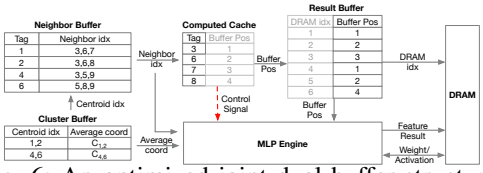


Fig. 6: An optimized joint dual buffer structure.

the PRADA algorithm is inserted into the feature computation. In PRADA, we first apply the front-end advanced re-clustering and generate cluster information. For instance, we get the centroid point indices (P_1, P_2) and (P_4, P_6), which respectively belong to the first and second clusters. And we calculate the average coordinates $\mathbf{C}_{1,2}$ and $\mathbf{C}_{4,6}$ of the two clusters. Afterwards, we execute the back-end dynamic approximation to make the input vectors that are in the same cluster equivalent by substituting the coordinates of the centroid points P_1 and P_2, $\mathbf{C}_1$ and $\mathbf{C}_2$, with the average coordinate $\mathbf{C}_{1,2}$. Then, the 12×6 LPMatrix can be compressed by squeezing the same input vectors out. Consequently, the output feature can be efficiently computed by feeding the squeezed 8×6 LPMatrix into MLP so as to obtain an 8×128 OFMatrix, where 128 is the output feature dimension. In the end, we reconstruct the 12×128 OFMatrix in a copy-and-paste manner. For example, we reconstruct M($\{\mathbf{C}_3$-$\mathbf{C}_2,\mathbf{F}_3\}$) by copying M($\{\mathbf{C}_3$-$\mathbf{C}_{1,2},\mathbf{F}_3\}$). In this example, we lower the computation complexity of MLP by reducing the input vectors from 12 to 8.

IV. Proposed Architecture

The proposed PRADA architecture is sketched in Fig. 5. It consists of front-end (data preparation) and back-end (PRADA algorithm execution). The front-end consists of a cluster buffer and neighbor buffer, which prepares the cluster, centroid index, and neighbor index by communicating with DRAM. The back-end mainly includes an approximation engine for dynamically approximating the input vectors, an MLP engine for calculating the output features of points, and the associated control logic.

A. The Back-end Approximation Engine

1) An Intuitive Solution: Because PRADA saves the feature computations by dynamic approximation and feature recon-

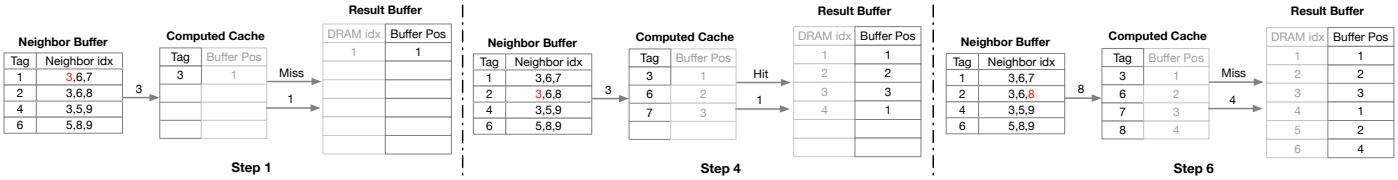


Fig. 7: The joint dual buffer building process.

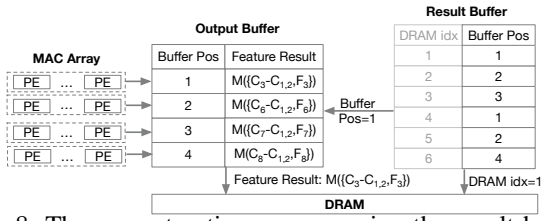


Fig. 8: The reconstruction process using the result buffer.

struction, it is importance to reuse the computed feature results as much as possible. So, an intuitive architecture is to apply a cache in the approximation engine with the following steps.

1) In each step, each neighbor point probes the cache. The cache will regard the neighbor index as a tag to check if there exists any similar input vector that has already been calculated before. If the cache hits, the corresponding feature computation can be skipped, and the feature result in the hit entry will be directly written back to reconstruct the OFMatrix.

2) If the cache misses, the feature computation works as normal without any savings, and the corresponding feature results will be stored in the cache for later reuse.

A better cache should identify all the similarities by storing all features in one cluster on-chip. However, the feature result is too large in size to hold all of them (up to 96KByte in PointNet++ [10]) in the cache.

2) Joint Dual Buffer Structure: Instead, we should immediately consume the feature results instead of storing them on-chip once the MLP engine generates the feature results. Consequently, we propose the joint dual buffer, including the result buffer and computed cache, to substitute the intuitive cache as illustrated in Fig. 6. The result buffer records the position of the feature result in the output buffer in the MLP engine. The computed cache records the neighbor index if it has been calculated before. In this way, an entry in the computed cache will serve as a reference point that helps PRADA to identify the similarity dynamically.

Fig. 7 illustrates how to build the joint dual buffer. We assume two centroid points P_1 and P_2 are in the same cluster after advanced re-clustering. And each point has three neighbor points P_3, P_6, P_7 and P_3, P_6, P_8.

In step 1-3, because P_3, P_6, P_7 to centroid P_1 first appear in the computed cache, these input vectors $I_{3,1}$, $I_{6,1}$, $I_{7,1}$ will be computed by MLP engine. The feature results will be stored in the 1^{st}, 2^{nd}, 3^{rd} positions in the output buffer. The computed cache stores 3, 6, and 7 as tags, and the result buffer holds 1, 2, and 3 as output buffer positions.

In step 4, P_3 comes again belonging to another centroid point P_2, the feature will be approximated, and whose results will be reconstructed. The computed cache will capture P_3's recurrence and use "1" indicating the output buffer position to fill the result buffer at the 4^{th} entry because the DRAM address increments linearly to the incoming points.

In step 6, because P_8 first appears, the computed cache and result buffer act similarly to steps 1-3.

After the MLP computation, the result buffer then guides the feature reconstruction of OFMatrix and writes it back to DRAM with the assistance of the output buffer in the MLP engine. As shown in Fig. 8, the output buffer in MLP engine now holds four feature results whose neighbor points are P_3, P_6, P_7, P_8. The result buffer will read them out in order as recorded in the "Buffer Pos" field and write them back in a DRAM burst operation.

The computed cache can capture all the similarities with enough entries and record the indices instead of data to shrink its size. The result buffer streamlines the DRAM accesses with awareness of the dynamic approximation opportunity. In PRADA architecture, they succeed in eliminating all the unnecessary computation and memory accesses by working together with minimum on-chip storage. Although they take sequential steps in filling their entries, the time can totally overlap with the time-consuming MLP computation.

V. Experimental Methodology

A. Validation on Accuracy

We evaluate our PRADA algorithm by applying Pytorch [8] framework. The datasets we used are: 1) ModelNet40 [14]; 2) ShapeNet [2]; 3) S3DIS [1], which are well-acknowledged in various point cloud algorithms. And we compare the accuracy of PRADA with PointNet++ [10] (denote as "baseline" in the figures) and Mesorasi [5].

B. Modeling Accelerator Architecture

To evaluate the performance of PRADA, we developed a cycle-accurate performance model with 256 GB/s HBM2 bandwidth. The design is implemented in RTL and synthesized by Synopsys Design Compiler to get the area and power consumption under 45nm technology, with a design frequency at 500MHz. We use CACTI [7] to model the SRAM and DRAM. The total area of the PRADA architecture is 5.2 mm^2, and the MLP engine takes up most of the area. Concretely, the computed cache and result buffer in the approximation engine take 2KByte. And the neighbor buffer and cluster buffer cost 1KByte and 2KByte. Lastly, the MLP engine consists of a 16×32 MAC array and 98 KByte on-chip buffer.

We compare PRADA architecture with modern hardware platforms, including server GPUs (NVIDIA Tesla V100 PCIe 32GB, NVIDIA GTX 1080 Ti), server CPU (Intel(R) Xeon(R)

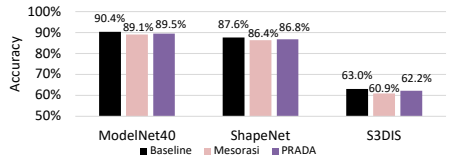


Fig. 9: Accuracy of PointNet++, Mesorasi and PRADA.

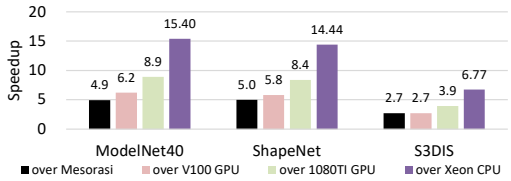


Fig. 10: Speedup over GPU and CPU.

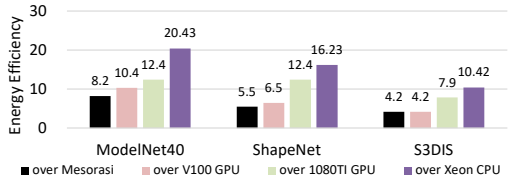


Fig. 11: Energy efficiency over GPU and CPU.

CPU E5-2609 v4 @ 1.7GHz). We also compare to the state-of-the-art point cloud accelerators Mesorasi [5] and PointAcc [6].

VI. Experimental Results

A. Accuracy

Fig. 9 compares our PRADA to the baseline PointNet++ and Mesorasi on three datasets. For ModelNet40 dataset, PRADA can match the baseline's accuracy with a loss of less than 0.9%. Moreover, PRADA has an accuracy loss of less than 0.8% for ShapeNet and S3DIS datasets. PRADA may incur accuracy loss for some cases, e.g., the dataset that contains a mass of objects in the environment, so the advanced re-clustering operation cannot precisely cluster the centroid points. Fig. 9 also compares the accuracy of Mesorasi and ours. We can see that PRADA is 0.4% better than Mesorasi for ModelNet40 dataset. In particular, PRADA is 1.3% better for S3DIS dataset as the S3DIS dataset is more complicated, and Mesorasi cannot handle it by roughly applying NN on raw points. The accuracy degradation implies its straightforward solution is unacceptable, particularly in intricate benchmarks. In contrast, PRADA reserves the accuracy with improved performance.

B. Performance Results

Fig. 10 shows the speedup of PRADA over Mesorasi, V100 GPU, 1080TI GPU, and Xeon CPU on three tasks. Since the benefit of our scheme has a close relationship with the number of approximated input vectors and the number of centroid points in one cluster, the performance of our scheme varies across different benchmarks for balancing accuracy and performance. For example, for ModelNet40 dataset, PRADA achieves nearly $4.9\times$, $6.2\times$, $8.9\times$, and $15.4\times$ speedup over Mesorasi, V100 GPU, 1080TI GPU, and Xeon CPU. And averagely, PRADA can achieve $4.2\times$, $4.9\times$, $7.1\times$, and $12.2\times$ speedup over Mesorasi, V100 GPU, 1080TI GPU, and Xeon CPU. This is because: 1) PRADA can reduce a significant number of input vectors passing through the large MLP computations. It greatly simplifies the processing for the approximated input vectors with a much reduced reconstruction operation; 2) PRADA architecture introduces an approximation engine that captures the similarities while eliminating all the unnecessary memory accesses; 3) Mesorasi can only accelerate the first layer as the accumulated non-linearity MLP introduces severe accuracy loss if deploying it to all layers; 4) The limited matrix size make GPU underutilized while PRADA processes 16×32 operations in parallel so that PRADA is easy to be fully utilized.

TABLE II: Comparison with existing PCR accelerators.

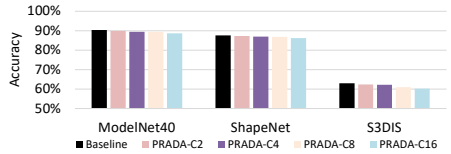
	Mesorasi	PointAcc	PRADA	PRADA+ PointAcc
Acceleration Technique	Software+ Hardware	Hardware	Software+ Hardware	Software+ Hardware
Technology	16nm	40nm	45nm	45nm
Effective Throughput	512GOP/s	922GOP/s	2.2TOP/s	4.4TOP/s

Fig. 11 shows the energy efficiency results. Overall, PRADA is $6.0\times$, $7.0\times$, $10.9\times$, and $15.7\times$ better than Mesorasi, V100 GPU, 1080TI GPU, and Xeon CPU respectively. Energy savings come from our specialized PRADA architecture, which can reduce unnecessary input vectors access from/to the off-chip memory and on-chip buffers.

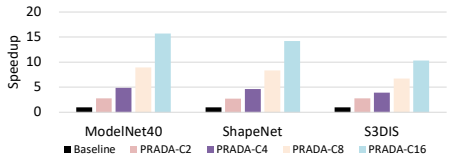
Table II compares PRADA with the state-of-the-art PCR accelerators Mesorasi [5] and PointAcc [6]. Mesorasi applies a delayed-aggregation that only works in the first layer, leading to a relatively low computation saving. PointAcc is a pure hardware accelerator for PCR. Since PointAcc supports the FPS and neighbor search by designing the specialized module, PointAcc can improve the performance. Overall, it can achieve $1.8\times$ speedup over Mesorasi. Moreover, PRADA achieves $4.2\times$ speedup compared to Mesorasi because PRADA applies a novel dynamic approximation that greatly reduces the input vectors being executed by the large MLP computations. Last but not least, PRADA is orthogonal to PointAcc so that we can distribute the FPS and neighbor search operations into PointAcc while performing the feature computation with the dynamic approximation in PRADA architecture. In this way, PRADA+PointAcc can significantly improve the performance by $8.6\times$ compared to Mesorasi. Specifically, assuming under 500MHz and 512 MACs (16×32), Mesorasi's throughput is 512GFLOPS. Since PointAcc, PRADA, and PRADA+PointAcc respectively have $1.8\times$, $4.2\times$ and $8.6\times$ geomean speedup respectively, their effective throughput are $1.8 \times 512 = 922$GFLOPS, $4.2 \times 512 = 2.2$TFLOPS, and $8.6 \times 512 = 4.4$TFLOPS.

C. Detailed Analysis

1) Exploring the number of centroid points in one cluster (C).: Fig. 12(a) and (b) explore the impact of the number of centroid points in one cluster (C) on accuracy and speedup over 1080TI GPU on the three datasets. The label "PRADA-C2" represents two centroid points in one cluster. From the plot, we find that the accuracy suffers more loss at larger C. Too many centroid points may result in inaccurate approximation. On the other hand, the performance improves with a larger number of centroid points in one cluster. This is understandable because the more centroid points are, the more likely we



(a)



(b)

Fig. 12: Accuracy and speedup of PRADA by varying the number of centroid points in one cluster C.

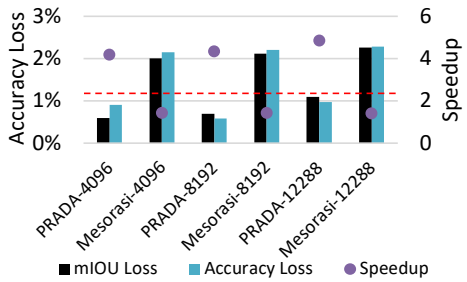


Fig. 13: Speedup and accuracy loss of Mesorasi and PRADA by varying the number of input points.

can leverage the advantage of our scheme for speeding up. Specifically, we find that PRADA can maintain the accuracy when $C = 8$ on ModelNet40 and ShapeNet datasets, while $C = 4$ on S3DIS dataset, because S3DIS is more complicated, so smaller re-clustering should be used to lean for accuracy. Fortunately, the re-clustering can be dynamically adjusted in PRADA architecture.

2) Exploring the number of input points.: Fig. 13 compares the speedup over 1080TI GPU and accuracy loss of PRADA and Mesorasi [5] in S3DIS dataset by varying the number of input points. "PRADA-4096" denotes the number of input points is 4096 for PRADA method. From the plot, we can see that as the number of input points increases, the speedup of Mesorasi remains while the accuracy loss becomes higher. This is because Mesorasi roughly operates on the entire raw input points and ignores the key features of local pairs. On the contrary, PRADA is good at handling a larger input point cloud by dynamically approximating the local pairs, given that there can be more adjacent centroid points with shorter distance for approximation. This benefits both accuracy and performance.

VII. Related Works

Point Cloud Recognition. Recently, PCR has received tremendous interest from the research groups. There are two major categories. One is to explore the data representation for high accuracy, such as voxelization [14] and raw points [9], [10]. However, such techniques require a large amount of computing time.

Other is to search for a new way to improve the efficiency of PCR. For example, Mesorasi [5] statically applies DNN on raw input points so that it can avoid executing feature computation on local pairs with massive redundancy. But Mesorasi has two drawbacks: 1) once the input points are too much, Mesorasi cannot achieve considerable speedup; 2) Mesorasi sacrifices the PCR accuracy because it ignores the local structures.

DNN Accelerators. In the last few years, various DNN accelerators [4], [12], [11], [13] are proposed. For example, Diannao series [4] introduce multiple NN accelerators by integrating a large on-chip eDRAM. Since reducing DRAM accesses is important to performance, PRADA architecture also benefits from these designs.

In fact, DNNs are hard to fit the on-chip memory, researchers such as Eyeriss [3] explores dataflow to improve data reuse. PRADA can work with the dataflow optimization. PRADA steps forward by closely interacting with the algorithm, and leverages the MAC array as an efficient infrastructure for dynamic approximation.

VIII. Conclusion

This paper first offers a new PRADA algorithm to dynamically approximate the input vectors based on the same features and same clusters characteristics for neighbor and centroid points respectively. Moreover, we propose the PRADA architecture to efficiently implement the proposed PRADA algorithm. Our evaluation shows that the proposed PRADA scheme outperforms other similar schemes in performance and accuracy.

References

[1] I. Armeni et al, "3d semantic parsing of large-scale indoor spaces," in *CVPR*, 2016, pp. 1534–1543.
[2] A. X. Chang et al, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.
[3] Y.-H. Chen et al, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.
[4] Y. Chen et al, "Dadiannao: A machine-learning supercomputer," in *MICRO*. IEEE, 2014, pp. 609–622.
[5] Y. Feng et al, "Mesorasi: Architecture support for point cloud analytics via delayed-aggregation," in *MICRO*. IEEE, 2020, pp. 1037–1050.
[6] Y. Lin, Z. Zhang, H. Tang, H. Wang, and S. Han, "Pointacc: Efficient point cloud accelerator," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 449–461.
[7] N. Muralimanohar et al, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.
[8] A. Paszke et al, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, pp. 8026–8037, 2019.
[9] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *CVPR*, 2017, pp. 652–660.
[10] C. R. Qi et al, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *NIPS*, vol. 30, 2017.
[11] Z. Song, B. Fu, F. Wu, Z. Jiang, L. Jiang, N. Jing, and X. Liang, "Drq: dynamic region-based quantization for deep neural network acceleration," in *ISCA*. IEEE, 2020, pp. 1010–1021.
[12] Z. Song, J. Wang, T. Li, L. Jiang, J. Ke, X. Liang, and N. Jing, "Gpnpu: enabling efficient hardware-based direct convolution with multi-precision support in gpu tensor cores," in *DAC*. IEEE, 2020, pp. 1–6.
[13] Z. Song, F. Wu, X. Liu, J. Ke, N. Jing, and X. Liang, "Vr-dann: Real-time video recognition via decoder-assisted neural network acceleration," in *MICRO*. IEEE, 2020, pp. 698–710.
[14] Z. Wu et al, "3d shapenets: A deep representation for volumetric shapes," in *CVPR*, 2015, pp. 1912–1920.