An Energy-Efficient 3D Point Cloud Neural Network Accelerator With Efficient Filter Pruning, MLP Fusion, and Dual-Stream Sampling

[1]Changchun Zhou, [1]Yuzhe Fu, [1]Min Liu, [1]Siyuan Qiu, [1]Ge Li, [2]Yifan He, and [1]Hailong Jiao

[1]School of Electronic and Computer Engineering, Shenzhen Graduate School, Peking University, Shenzhen, China
[2]Reconova Technologies Co., Ltd., Xiamen, China

Email: jiaohailong@pku.edu.cn

Abstract—**Three-dimensional (3D) point cloud has been employed in a wide range of applications recently. As a powerful weapon for point cloud analysis, point-based point cloud neural networks (PNNs) have demonstrated superior performance with less computation complexity and parameters, compared to sparse 3D convolution-based networks and graph-based convolutional neural networks. However, point-based PNNs still suffer from high computational redundancy, large off-chip memory access, and low parallelism in hardware implementation, thereby hindering the applications on edge devices. In this paper, to address these challenges, an energy-efficient 3D point cloud neural network accelerator is proposed for on-chip edge computing. An efficient filter pruning scheme is used to skip the redundant convolution of pruned filters and zero-value feature channels. A block-wise multi-layer perceptron (MLP) fusion method is proposed to increase the on-chip reuse of features, thereby reducing off-chip memory access. A dual-stream blocking technique is proposed for higher parallelism while maintaining inference accuracy. Implemented in an industrial 28-nm CMOS technology, the proposed accelerator achieves an effective energy efficiency of 12.65 TOPS/W and 0.13 mJ/frame energy consumption for PointNeXt-S at 100 MHz, 0.9 V supply voltage, and 8-bit data width. Compared to the state-of-the-art point cloud neural network accelerators, the proposed accelerator enhances the energy efficiency by up to 66.6× and reduces the energy consumption per frame by up to 70.2×.**

Keywords—Three-dimensional point cloud, off-chip memory access, data reuse, high parallelism, speedup, energy efficiency.

I. Introduction

In recent years, point cloud is increasingly used in various applications, such as cars, robots, drones, and depth cameras in smartphones. Point cloud demonstrates significant advantages such as easy acquisition, high resolution, and efficient data format. Therefore, point cloud has become an important modality as important as images and videos for deep learning applications such as autonomous driving, photography, virtual reality (VR), and augmented reality (AR). Deep neural networks (DNNs) have been utilized for point cloud analysis. The mainstream point cloud neural networks (PNNs) can be grouped into three categories: sparse 3D convolution-based networks [1], [2], [3], graph-based convolutional neural networks [4], [5], [6], [7], [8], and point-based PNNs [9], [10], [11], [12], [13]. In particular, point-based PNNs, a class of neural networks developed from PointNet [9], perform outstandingly in point cloud analysis. The point-based PNNs can directly process points to extract features and solve the issues of disordered and discrete data. Furthermore, the mapping operations (including sampling and grouping) in point-based PNNs enable the networks to have multi-level and strong feature extraction capability, thereby achieving high inference accuracy and robustness. The use of shared-weight MLP and pooling in point-based PNNs makes the models compact with light computation.

When deploying PNNs on edge devices, such as wearable, mobile, and Internet-of-Things (IoT) devices, which require real-time interactions with humans and instantaneous perception and comprehension of the environment, both high inference accuracy and minimal latency/energy consumption are crucial. However, the grouping operations in point-based PNNs, such as k-nearest neighbors (KNN) and ball query (BQ), lead to irregular indexing patterns. These irregular patterns degrade the off-chip memory access bandwidth efficiency significantly. The limited data reuse in the MLP layers of PNNs also results in substantial off-chip memory access, which is exacerbated by the large amount of computation and parameters in PNNs. Furthermore, the commonly used farthest point sampling (FPS) algorithm is inherently sequential and becomes a latency bottleneck in PNN accelerators. Therefore, there is an urgent need to design a point cloud neural network accelerator that minimizes off-chip memory access and enhances computation parallelism of the sampling operations, while reducing the amount of computations.

On-chip PNN accelerators have been investigated sporadically in recent years. For sparse 3D convolution-based PNNs, in [14], a customized skipping index rule table and an efficient search method are proposed for accelerating MinkowskiNet [2] to reduce the hardware overhead for storage management. However, [14] suffers from high energy consumption caused by enormous computation and movement of index. In [15], an accelerator for SECOND network [1] is proposed. The coordinate (COO) and feature map are block-partitioned according to the COO, eliminating the overhead of off-chip gathering and scattering, and enabling efficient continuous block-level off-chip memory access. However, [15] suffers from significant repetitive off-chip memory access and on-chip computation for the boundary voxels between blocks.

For graph-based PNNs, Point-X [16] extracts and exploits the spatial locality in point cloud data for efficient processing DGCNN [4]. Point-X clusters the point cloud and assigns each cluster to a compute tile to maximize intra-tile computational parallelism and minimize inter-tile data movement. However, Point-X is highly customized for DGCNN and lacks the generality for other PNNs.

For point-based PNNs, a delayed-aggregation method is proposed in Mesorasi [17] to increase the reuse of point features. Delayed-aggregation moves aggregation from before multiple MLPs to after multiple MLPs in point-based PNNs. However, the delayed-aggregation method reverses the computation order of centralization and activation of features, resulting in unstable inference accuracy and up to 0.9% accuracy loss. Furthermore, the number of the output feature map channels often doubles compared to that of input feature maps per MLP, making the delayed aggregation a new latency bottleneck beyond MLPs. Evolved from Mesorasi, Crescent [18] focuses on eliminating the irregular memory access of features and coordinates. An approximate neighbor search method is proposed to reduce the data movement by locally searching neighbor points with Crescent. Meanwhile, the access conflicts of memory banks are selectively eliminated by copying the access data to accommodate multiple port requests. However, up to 1% accuracy loss is caused by the two approximation techniques. In [19], PNNPU is proposed to accelerate VoteNet [20]. Point cloud is mapped and convolved in blocks to reduce the complexity of mapping operations and eliminate the irregular off-chip memory access of MLPs. However, the proposed blocking method suffers from a significant accuracy loss in small-scale scenarios (e.g., 1k input points). Furthermore, the large number of computations and off-chip memory access involved in the convolutions of PNNPU still remain as great challenges. A configurable PNN accelerator, PointAcc, is proposed in [21] to support multiple types of PNNs. However, the off-chip memory access of coordinates and features is large, while the reuse rate of the mapping hardware is low due to the configurability of the accelerator.

To address the existing challenges, including large computational complexity, off-chip memory access, accuracy loss, and low parallelism, an energy-efficient and low-latency PNN accelerator is proposed for real-time 3D point cloud inference on edge devices. This accelerator supports the mainstream point-based PNNs, e.g. PointNet [9], PointNet++ [10], PointMLP [11], and PointNeXt [12]. An efficient filter pruning is applied to skip the redundant convolution of pruned filters and zero-value feature channels. Meanwhile, a large amount of inefficient off-chip memory access of grouping point features is eliminated. A block-wise multi-layer perceptron (MLP) fusion method is proposed to improve the on-chip reuse of features and reduce off-chip memory access of point features in MLPs. To eliminate the latency bottleneck of sampling, a dual-stream blocking technique is proposed for higher parallelism of sampling while maintaining inference accuracy. Specialized architecture and computation flow are proposed to enhance the effectiveness of the proposed techniques. Implemented in the TSMC 28-nm CMOS technology, the proposed accelerator achieves a remarkable effective energy efficiency of 12.65 TOPS/W and 0.13 mJ/frame energy consumption for PointNeXt-S at 100 MHz, 0.9 V supply voltage, and 8-bit data width. Compared to the state-of-the-art PNN accelerators, the proposed accelerator enhances the energy efficiency by up to 66.6×, reduces the energy consumption by up to 70.2×, and improves the overall efficiency (Frame/mm^2/mJ) by up to 127.5×.

The paper is organized as follows. The background of point-based PNNs is introduced in Section II. The system architecture is portraited in Section III. The proposed techniques that enable low latency, high parallelism, and minimal off-chip memory access are presented in Section IV. The experimental results of the proposed PNN accelerator are discussed in Section V. The paper is concluded in Section VI.

II. Background

In this section, the fundamental concepts of point cloud, mapping operation, and MLP are provided. The backbone of point-based PNNs and a classical point-based PNN, PointNeXt-S, are introduced.

A. The Backbone of Point-based PNNs

The point cloud is a set of points $\boldsymbol{x} = \{\boldsymbol{x}_k\} = \{(\boldsymbol{p}_k, \boldsymbol{f}_k)\}$, where $\boldsymbol{p}_k = (\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{z}_k)$ is the coordinate of the k^{th} point, and $\boldsymbol{f}_k$ is the corresponding 1-D feature vector. As shown in Fig. 1c, the backbone of point-based point cloud neural network architecture is composed of multiple stages. Each stage contains a mapping layer, several consecutive MLPs, and a pooling layer. The mapping layer contains sampling and grouping operations and aims to build a relationship between the input and output points as a mapping. These operations usually only take point coordinates as input. Then, the corresponding features of input points are grouped according to the mapping. The convolutions in the MLPs are performed on all the feature groups. Then, max-pooling is performed on the output results of each group to obtain the corresponding output point feature.

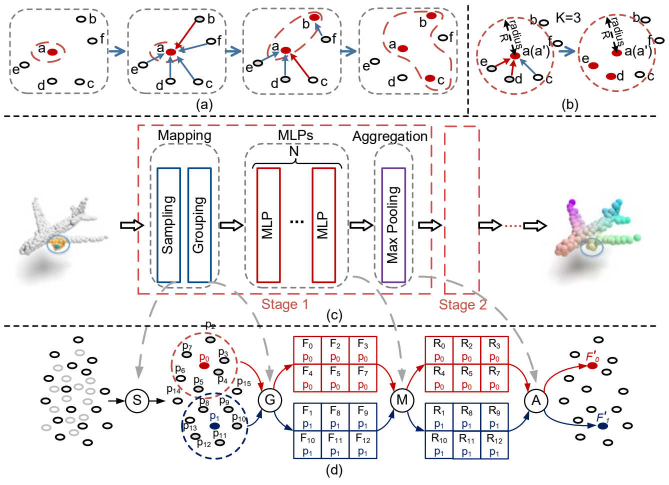

Fig. 1. The backbone of point-based PNNs. (a) FPS operation process. (b) BQ operation process. (c) The backbone overview of point-based PNNs. (d) The change of points within a stage.

Sampling. Point-based PNNs typically employ the farthest point sampling (FPS) algorithm for the sampling process, where output points are iteratively sampled from the input point cloud. FPS starts by randomly selecting an initial point from the point cloud. In each iteration, the point that has the largest distance to the current output point cloud is added to the output point cloud. For example, in Fig. 1a, the first selected output point is assumed to be $\boldsymbol{a}$. Since $\boldsymbol{b}$ has the largest distance from $\boldsymbol{a}$, $\boldsymbol{b}$ is chosen as the second output point.

Grouping. After sampling, grouping refers to the process of finding the nearest neighbor points for each point of the output point cloud. Note that the coordinates of the input and output points remain unchanged, i.e., the coordinates of input point $\boldsymbol{a}$ and output point $\boldsymbol{a'}$ are the same. Point-based PNNs typically employ KNN or BQ for the grouping process. In KNN, the top-k nearest neighbor points are selected based on their distances to the output point. BQ further requires these neighbor points to lie within a sphere of radius $\boldsymbol{R}$. As illustrated in Fig. 1b, $\boldsymbol{K}$, the number of the required nearest neighbor input points for the output point $\boldsymbol{a'}$, is three. Four input points $\boldsymbol{a}$, $\boldsymbol{c}$, $\boldsymbol{d}$, and $\boldsymbol{e}$ are identified within the sphere centered at the output point $\boldsymbol{a'}$. BQ then selects the three nearest points ***a***, ***d***, and ***e***. Then, points ***a***, ***d***, and ***e*** are considered as the group of the output point ***a'***. The relationship between the output point ***a'*** and input points ***a***, ***d***, and ***e*** is the mapping of output point ***a'***. The corresponding features of points ***a***, ***d***, and ***e*** are also grouped and then passed to the following MLPs to generate the feature of output point ***a'***.

MLPs. MLP is a kind of convolution layer with a kernel size of 1×1. In the MLPs of point-based PNNs, filters are shared across all input point features. The change of points within a stage is shown in Fig. 1d, where ***p*** represents the coordinates of input points, ***F*** represents the grouped features, and ***R*** represents the partial sums after MLPs. MLPs are performed on the grouped input features to obtain partial sums. Multiple MLPs are stacked in each stage to further extract features.

Max-Pooling. In point-based PNNs, max-pooling is employed to aggregate the per-point partial sums obtained by MLPs from all neighbor points. For example, the channel-wise maximum values are computed among the partial sums $\boldsymbol{R_0}$, $\boldsymbol{R_2}$, $\boldsymbol{R_3}$, $\boldsymbol{R_4}$, $\boldsymbol{R_5}$, and $\boldsymbol{R_7}$ to obtain the output point feature $\boldsymbol{F_0'}$, as shown in Fig. 1d.

B. The Network Architecture of PointNeXt-S

PointNeXt is a state-of-the-art and scalable point-based PNN for processing point cloud and significantly outperforms other PNNs. Optimized from the classical PointNet++ [10], PointNeXt-S, a small variant of PointNeXt, introduces residual architecture into PointNeXt and mainly consists of five stages, as shown in Fig. 2. PointNeXt-S outperforms PointNet++ while maintaining similar amount of model parameters and computational complexity.

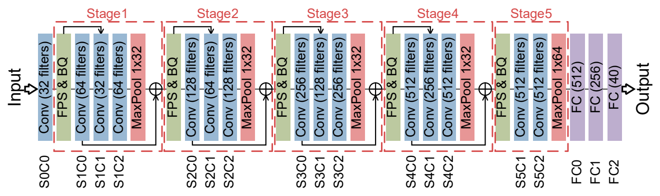

Fig. 2. The network architecture of PointNeXt-S.

To train and compress neural networks, we use Distiller [22], an open-source Python package for neural network compression research that is developed by Intel. ModelNet40 [23], a classical 3D point cloud dataset of 40 object classes and 12311 samples, is typically applied for point cloud neural network evaluation. Therefore, ModelNet40 is used as the benchmark for classification with PointNeXt-S.

III. System Architecture Overview

The system architecture of the proposed point cloud accelerator is introduced in this section. To obtain high energy efficiency and low latency, the hardware architecture is optimized for high data reuse, high parallelism, redundant operation skipping, as well as network configurability.

The hardware architecture of the proposed accelerator is shown in Fig. 3. The accelerator consists of a control module, a mapping module, a convolution module, a pooling module, a memory module, and an interface module. The mapping module includes a sampling submodule performing FPS and a grouping submodule performing KNN or BQ. The mapping module can be configured to support different numbers of input points and output points. The sampling submodule contains 16 cores, while the grouping submodule contains 8 cores. The convolution module is composed of 4 cores, each of which is a systolic array of 16×16 processing elements (PEs). The pooling module contains 8 cores, each of which consists of 32 comparators. The number of cores in each module is determined by the workload and latency requirements of each type of operations in point-based PNNs. The memory module is composed of 32 static random-access memory (SRAM) banks of 4KB. Due to the varied bandwidth and storage requirements of the mapping module, the convolution module, and the pooling module during the inference, the memory module can be dynamically allocated for these modules.

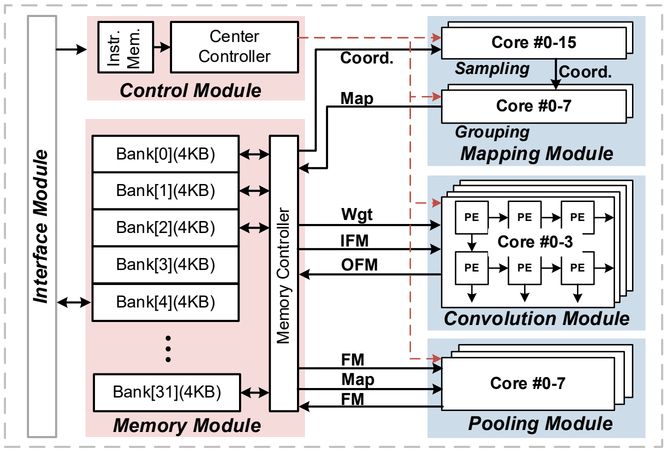

Fig. 3. The hardware architecture of the proposed point cloud neural network accelerator.

An instruction set architecture (ISA) is designed for the off-chip host to control the accelerator. During neural network inference, the off-chip host sends a set of instructions to the control module. The control module decodes the instruction and then controls the execution of the mapping module, the convolution module, and the pooling module, as well as the allocation of the memory module. As mentioned in Section II-A, the mapping module constructs the mappings between the input points and the output points. Then, the required input point features are grouped for each output point according to the built mappings. The multiple cores of the mapping module provide sufficient performance to meet the throughput of mappings required by the convolution module. The dataflow of the convolution module is output stationary (OS) dataflow [24]. The convolution module convolves the input feature groups with filters to obtain partial sum groups. The pooling module obtains the output point features by performing max-pooling on the partial sum groups. The proposed accelerator adopts a pipelined dataflow from overall architecture to microarchitecture for higher hardware utilization. The multi-core schemes are employed for higher parallelism and bandwidth utilization.

IV. Optimization Techniques

In this section, three techniques are proposed to enhance the energy efficiency and shorten the latency of the PNN accelerator. Firstly, to reduce the high energy consumption and long latency caused by convolution, an efficient filter pruning is proposed. The redundant convolution of pruned filters and zero-value feature channels is skipped, while a large amount of inefficient off-chip memory access of grouping point features is eliminated. Secondly, a block-wise multi-layer perceptrons (MLP) fusion is proposed to increase the on-chip reuse of features and significantly reduce off-chip memory access of point features in MLPs. Thirdly, to eliminate the latency bottleneck of sampling, a dual-stream blocking method is proposed to enhance the parallelism of sampling while maintaining the inference accuracy.

A. *Efficient Filter Pruning (EFP)*

Similar to 2D-CNN, the convolution operations, including MLPs and FCs, still dominate the computation and data storage of point-based PNNs, as shown in Fig. 4a and Fig. 4b. High energy consumption and long latency are caused by the computation and off-chip memory access, respectively, involved in convolutions. Meanwhile, the grouping operations, such as k-nearest neighbors (KNN) and ball query (BQ) in point-based PNNs, lead to irregular off-chip memory access of point features. Therefore, the off-chip memory access bandwidth efficiency of PNNs is significantly degraded. An efficient model compression method to accommodate more point features on chip is desirable. Prior works [25] have shown that significant redundancy exists in 2D-CNN, so that a large number of weights and activations can be set to zero without accuracy loss, which is also verified in this work for point-based PNNs. In a convolution layer, element sparsity is the ratio of zero-value elements in weights or activations, while channel sparsity is the ratio of zero-value channels in filters or input features. All elements are zero in a zero-value channel. Including these zeros increases data storage and wastes energy and computation time. Skipping these zeros in computation does not impact the network accuracy at all. Therefore, effectively skipping zero channels can significantly reduce the inference time and energy consumption of the accelerator.

After linear quantization to fixed-point 8-bit integers, the element sparsity of original weights and activations in PointNeXt-S is 2.07% and 41.75%, respectively. The element sparsity is low and unstructured, not friendly for hardware acceleration. An efficient and structured pruning method is desirable. Since the contribution of different filters to the network accuracy varies greatly, some less important filters can be directly pruned without accuracy loss. Meanwhile, note that in the MLPs of point-based PNNs, the filters are reused in only one dimension across input points, rather than in two dimensions such as 2D-CNN. Therefore, in point-based PNNs, a greater number of filters can be pruned with minimal impact on the output features and inference results compared to 2D-CNN. An efficient filter pruning (*EFP*) is proposed in this work. The overall pruning rate of filters is set to 78%. The pruning rate of each layer is determined based on the sensitivity analysis by Distiller. After filter pruning, the channel sparsity of features and filters for each layer is significantly improved, as shown in Fig. 4c. Since the batch normalization among MLPs is removed in PointNeXt-S, the pruned filters in the current layer can also introduce the corresponding zero-value channels of the input feature in the next layer, increasing the overall channel sparsity of features from 15.7% to 61.78%. Meanwhile, the convolution of the zero-value channels of input features and the corresponding channels of the filters in the next layer is redundant. Consequently, these zero-value channels in the filters of the next layer can be further pruned, improving the overall channel sparsity of filters from 78% (the overall pruning rate of filters) to 96.6%. The OS dataflow of the convolution module provides strong support for structured pruning, resulting in minimal hardware overhead. After applying *EFP* in PointNeXt-S, the model parameters and computational complexity are significantly reduced by 29.4× and 12.4×, respectively, as shown in Fig. 4d. Furthermore, the input point features of each stage can be stored on-chip because the zero-value channels of input point features introduced by *EFP* are not stored, thereby eliminating a large amount of inefficient off-chip memory access of grouping point features and significantly shortening the latency.

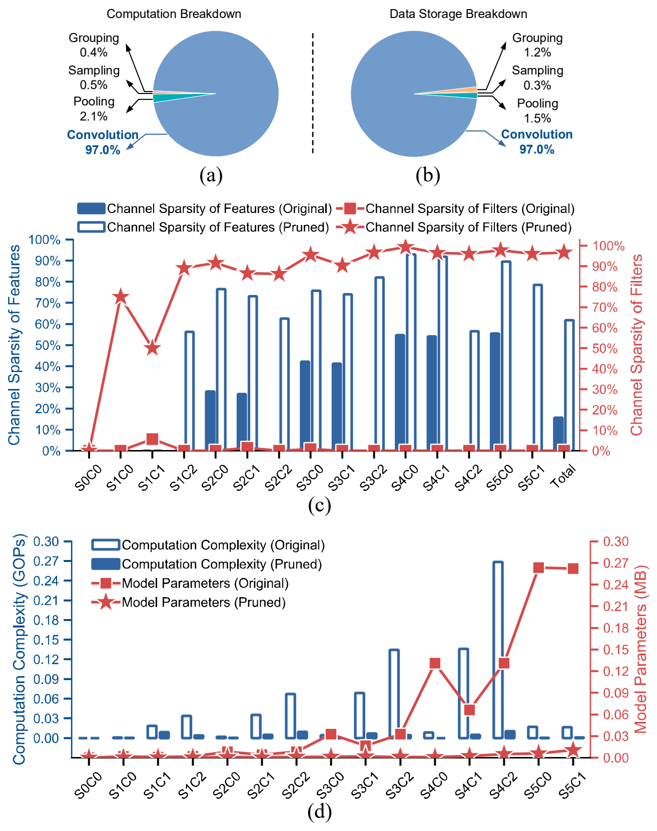

Fig. 4. Efficient filter pruning (*EFP*) in PointNeXt-S. (a) Computation and (b) data storage breakdown of each type of operations in PointNeXt-S. (c) Channel sparsity of input features and filters after the original PointNeXt-S is quantized to fixed-point 8-bit integers without pruning and after filters of the quantized PointNeXt-S are further pruned by 78%. (d) Reduction of model parameters and computational complexity after filters are pruned by 78%.

B. *Block-wise MLP Fusion (BMF)*

As shown in Fig. 1c, point-based PNNs are composed of multiple consecutive stages, each of which contains a series of MLPs. In terms of size, the filters are light in the shallow stages while heavy in the deep stages in the original point-based PNNs, such as PointNeXt-S, as shown in Fig. 5a. However, different from 2D-CNN, point-based PNNs contain grouping operations, as mentioned in Section II-A. As shown in Fig. 5c, the input point features of the current stage are grouped to generate the grouped input point features. Note that the size of the grouped point features is K× (e.g. 32× in PointNeXt-S) larger than the original input point features of the current stage. The grouped output point features are performed aggregation after the convolution of MLPs and thus reduced by K×. Namely, the point features among MLPs are dilated by $K\times$, compared to the point features among stages. Therefore, the size of features is larger than the size of filters in most stages. For example, in the shallow stages of the original PointNeXt-S, the features dominate the storage, reaching 32× the size of the filters, and therefore cannot be completely stored on chip. After *EFP*, the features are even 51.4× larger than the filters in the pruned PointNeXt-S.

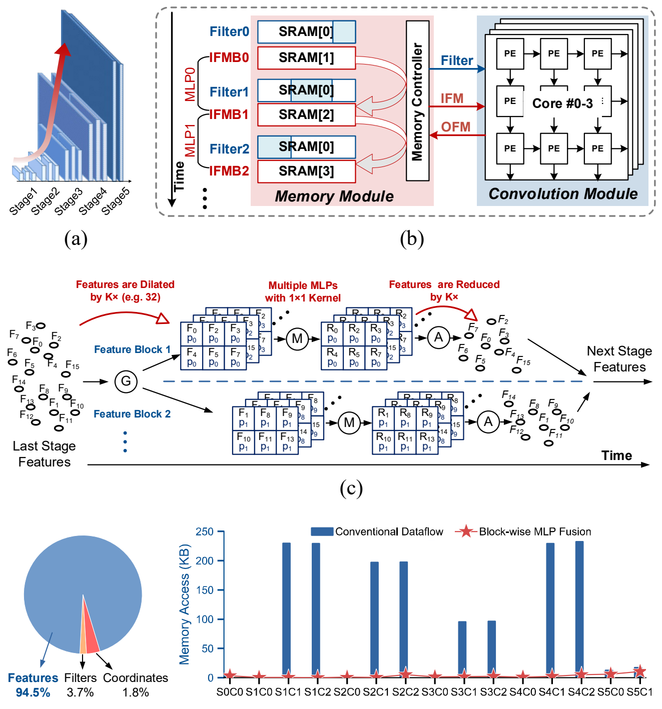

Fig. 5. Block-wise MLP fusion (*BMF*). (a) The size of filters in each stage of the original PointNeXt-S. (b) The hardware implementation of *BMF*. (c) The data flow of block-wise and pipelined MLP fusion. (d) The breakdown of off-chip memory access after *EFP* is applied. (e) The memory access reduction of each layer after *BMF*.

The on-chip reuse of point features in point-based PNNs is also significantly reduced due to the 1×1 kernel size of MLPs, compared to 2D-CNNs. For example, the reuse of features in PointNeXt-S is reduced by ~9×, compared to VGG-16 [26]. In the conventional data flow, after the grouped input point features of the current MLP are convolved, the grouped input point features of the next MLP begin to be computed. Due to these two reasons, namely the larger feature size and the lower on-chip reuse of features, the off-chip memory access of input and output features is significantly increased. As shown in Fig. 5d and Fig. 5e, in the conventional dataflow, the off-chip memory access is large, while the features dominate the off-chip memory access. Therefore, the limited interface bandwidth cannot meet the requirement of the convolution module and pooling module, causing low hardware utilization and long latency.

To address the large off-chip memory access caused by large grouped point features, block-wise MLP fusion (*BMF*) is proposed in this work. In *BMF*, multiple MLPs extract the grouped input point feature block on chip, as shown in Fig. 5c. In detail, the filters are pre-stored on-chip. Then a grouped input point feature block is generated by grouping the input point features of the current stage. The n^{th} grouped output point feature block is generated by convolving the n^{th} grouped input point feature block with the filters of the first MLP. The n^{th} grouped output point feature block is kept in the SRAM and is the n^{th} grouped input point feature block for the further convolution of the subsequent MLP. Thus, consecutive convolution of the n^{th} grouped point feature block across multiple MLPs is enabled without off-chip memory access of intermediate grouped point feature blocks. Once the n^{th} grouped point feature block is processed by multiple MLPs of the current stages, the $(n+1)^{th}$ grouped input point feature block starts to be convolved with the filters of the first MLP.

As shown in Fig. 5b, in hardware implementation, the filters of the MLPs (Filter0, Filter1, and Filter2) are cached in the memory module in advance via an instruction transferred by the off-chip host. A grouped input point feature block (IFMB0) and the filters of the first MLP (Filter0) are convolved in the convolution module to compute the output point feature block, which is then written back to the memory module. The grouped output point feature block is the grouped input point feature block (IFMB1) of the second MLP. Therefore, a feature block dataflow loop is built between the memory module and the convolution module, enabling the on-chip reuse of feature blocks across MLPs.

Two characteristics of point-based PNNs are exploited to enable the proposed *BMF*. Firstly, the structure of consecutive multiple MLPs in point-based PNNs enables the reuse of features across multiple MLPs. Secondly, the 1×1 kernel size of MLPs enables direct partitioning of the input point feature without complex padding processing. Meanwhile, *EFP* enables not only the filters in the shallow stages, but also the filters in the deep stages to be completely stored on-chip. Therefore, the feature blocks can be consecutively convolved on chip across multiple MLPs.

The effectiveness of *BMF* in reducing off-chip memory access is attributed to three aspects. Firstly, *EFP* reduces the size of filters, enables the filters to be completely stored on-chip, and thus extends the application of *BMF* from shallow stages to deep stages of point-based PNNs. Secondly, *EFP* significantly reduces the size of the input point features of each stage, thus enabling the input point features of each stage to be completely stored on-chip and eliminating the off-chip memory access. Thirdly, the grouped point features among MLPs are $K\times$ larger than the input point features among stages. *BMF* eliminates the off-chip memory access of the large grouped point features. The effectiveness of *BMF* is higher when there are more MLP layers and the grouped point features are larger. Overall, *BMF* enables the features to be reused on-chip and eliminates the off-chip memory access of the intermediate features, as shown in Fig. 5e. A 21.1× off-chip memory access reduction is achieved for PointNeXt-S with *BMF*. Benefiting from the dynamic allocation of the memory module for different sizes of filters and feature blocks across multiple MLP layers, and due to the absence of complex control logic and storage overhead caused by padding-free feature blocks, *BMF* incurs minimal hardware overhead. Meanwhile, the configurable convolution module supports different sizes of filters and feature blocks. After *EFP*, the weights of all MLP layers are light and can be cached in the memory module in advance to meet the bandwidth requirements for the convolution module. Grouping features, convolution, and pooling are therefore parallelized. Therefore, the hardware utilization of accelerator is highly increased by *BMF*.

C. Dual-Stream Block-wise FPS (DSBF)

Point-based PNNs typically employ the Farthest Point Sampling (FPS) algorithm for down-sampling to reduce the input point cloud size and enhance the representational capability by preserving contour points. As shown in Fig. 1a, in the original FPS, the computation of the next sampled point depends on the result of the current sampled points. Therefore, the original FPS processing is inherently sequential with a latency of $O(N^2)$ cycles and a high computational complexity of $O(N^2)$, where N represents the number of input points. The sequential operations make FPS a latency bottleneck in hardware implementation. For example, in the first layer of PointNeXt-S, where FPS consists of 1024 input points and needs to sample 512 points, the processing time of FPS is approximately 80× longer than the computation time of the systolic array. As shown in Fig. 6a, after *EFP* and *BMF* are applied, sampling (FPS) accounts for 88.8% latency of processing PointNeXt-S. Therefore, a method for FPS that supports parallel computing with low computational complexity and low latency is desirable.

A block-wise FPS algorithm is proposed in PNNPU [19] for reducing the latency of sampling. The block-wise FPS algorithm consists of two steps: predicting and block-wise FPS. Both steps employ two key parameters, which are sparsity coefficient S and the number of blocks B [19]. Larger values of S and B can further reduce the computational complexity, at the expense of severer degradation of network accuracy, as demonstrated in Fig. 6b. In the first step, a large sparsity coefficient S results in the loss of critical contour points with strong representational ability, leading to decreased accuracy in predicted numbers. In the second step, a larger block number B leads to fewer points per block. The error caused by a larger predicted number than the number of actual points significantly increases.

To enhance the accuracy while ensuring low computational complexity, a dual-stream block-wise FPS (*DSBF*) algorithm is proposed in this work, which also has two steps. In the first step, called dual-stream prediction, two distinct sparsity strategies are adopted to reduce the original input points into two small sparse point sets. Then, FPS is performed on each set. Afterwards, the remained points are partitioned into $B/2$ cubes. The points are counted for each cube. The counted numbers are multiplied by $S/2$ and then accumulated to obtain the final predicted number for each cube. Therefore, the predicted numbers are more accurate than PNNPU. In the second step, called dual-stream partition, the input point cloud is partitioned into $B/2$ cubes. Then, each cube is evenly sampled into two blocks to obtain total B blocks. Based on half of the predicted number in each cube from the first step, these blocks are performed FPS separately. Finally, all remained points in each block are aggregated to obtain the final output points.

An example of applying *DSBF* to a point cloud with 1024 points is shown in Fig. 6c. The sparsity factor S is set to 16. The points are divided into 16 blocks (B = 16). In the dual-stream prediction, two sets with 64 sparse points are obtained based on two sparse strategies (S_0 and S_1). After performing FPS and cube-wise counting on the remained points in each set, the numbers of remained points in the two sets are obtained. These two numbers are multiplied by 8 ($S/2$) and accumulated to obtain the final predicted number of the point cloud. The figure only shows the point distribution for the front 4 cubes, where the predicted numbers are 88, 56, 32, and 80, respectively. In the dual-stream partition, the original point cloud is first divided into 8 cubes. Taking the gray cube with 128 points as an example. The gray cube is further divided into two blocks by evenly sampling, each containing 64 points. Then the two blocks are performed FPS separately to obtain two output blocks, each of which contains 28 points. 28 is half of the predicted number 56 from the dual-stream prediction. Then, the two blocks are aggregated to obtain the final output cube with 56 remained points. Dual-stream partition can be performed in parallel in all 8 cubes for sampling all the output points.

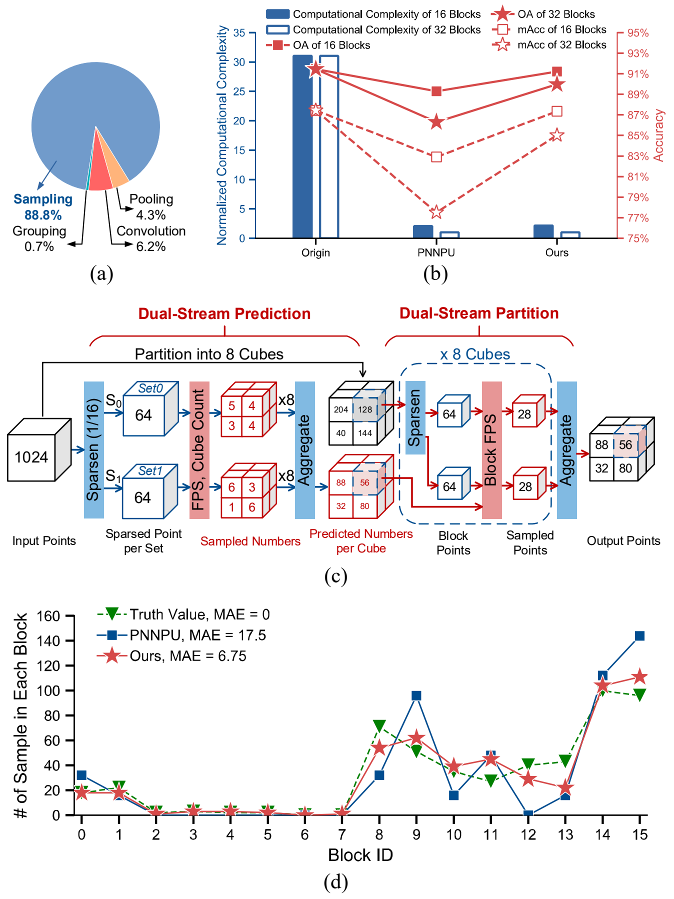

Fig. 6. Dual-stream block-wise FPS (*DSBF*). (a) Sampling dominates the overall latency after both *EFP* and *BMF* are applied. (b) Normalized computational complexity and accuracy of the original FPS, PNNPU, and the proposed *DSBF*. (c) The mechanism of *DSBF*. The '→' represents the dual-stream prediction data flow. The '→' represents the dual-stream partition data flow. Black numbers represent points, while red numbers show the number of points used for prediction. (d) The number of samples in each block.

A comparison of the predicted number of samples in each block of a point cloud among the original FPS (Truth Value), PNNPU, and the proposed *DSBF* is shown in Fig. 6d. The distribution of the proposed *DSBF* is similar to that of the original FPS. Alternatively, PNNPU shows significant errors compared to the original FPS. To evaluate the accuracy of different techniques quantitatively, the mean absolute error (MAE) is employed as a performance metric, using the original FPS as a reference. Our method achieves 2.59× reduction in

MAE compared to PNNPU. The normalized computational complexity and the accuracy of PointNeXt-S with the original FPS, PNNPU, and the proposed *DSBF* are shown in Fig. 6b. The example is with 1024 input points, 512 output points, and 16 or 32 blocks. In the case of 16 (32) blocks, the *DSBF* achieves a network overall accuracy (OA) of 91.21% (89.99%) and mean accuracy (mAcc) of 87.36% (85.02%), with 1.91% (3.69%) OA and 4.46% (7.51%) mAcc improvement, compared to PNNPU, and 0.24% (1.46%) OA and 0.07% (2.41%) mAcc loss, compared to the original FPS. Compared to the original FPS and PNNPU, the proposed *DSBF* saves 14.22× (30.1×) computational complexity and introduces only 5.8% (3.1%) computational complexity overhead, respectively. Therefore, the proposed *DSBF* significantly reduces the computational complexity while maintaining relatively high network accuracy.

V. Experimental Results

In this section, the simulation results of the proposed PNN accelerator are presented. The performance of the proposed accelerator is also compared with the state of the art.

A. Experiment Setup

The proposed accelerator is implemented in the TSMC 28-nm high-performance computing CMOS technology. Three types of standard cells (standard threshold voltage (SVT), high threshold voltage (HVT), and ultra-high threshold voltage (UHVT)) are used in the proposed accelerator. Cadence Genus is used for logic synthesis at 100 MHz and 0.9 V supply voltage. To evaluate the power consumption, Cadence NC-Sim is used to obtain the realistic switching activity of the hardware, in the form of TCF (toggle count format) files. The TCF files are then used for power evaluation with Cadence Genus. The PointNeXt-S network is trained in PyTorch and run as the benchmark on the proposed accelerator for evaluation. ModelNet40 is used as the evaluation dataset. The open-source tool Distiller is used for pruning, quantization to 8-bit data width, and retraining PointNeXt to recover the accuracy.

B. Performance Evaluation

The area and power breakdown of the proposed accelerator are shown in Fig. 7. The proposed accelerator can be configured to disable the three proposed techniques (*EFP*, *BMF*, and *DSBF*) to become a baseline version to evaluate the effectiveness of those techniques in detail. The improvement in speed and energy efficiency by using the three proposed techniques in PointNeXt-S at 0.9 V and 100 MHz is shown in Fig. 8.

As shown in Fig. 8, with the proposed *EFP* alone, the speedup and the effective energy efficiency improvement of the PNN accelerator are 1.07× and 5.51×, respectively, for PointNeXt-S, compared to the baseline. Note that the bottleneck of speed, when using *EFP* alone, is the FPS because the original FPS processing is inherently sequential with a latency of $O(N^2)$ cycles and a high computational complexity of $O(N^2)$. Therefore, *DSBF* is then used together with *EFP*. Compared to using *EFP* alone, the speed of the accelerator is boosted by 7.97×, meaning that the FPS algorithm is well accelerated in parallel by *DSBF*. Compared to the baseline, the energy efficiency of the accelerator is enhanced by 12.52×, by using *EFP* and *DSBF* together.

When using *EFP* and *DSBF* together, the bottleneck of speed is the large off-chip memory access of features. Thus, *BMF* is also applied to reuse features on-chip among MLPs afterwards, which achieves 21.1× reduction of total off-chip memory access. By combining *EFP*, *DSBF*, and *BMF* together, the speed and energy efficiency are enhanced by 1.8× and 1.1× respectively, compared to only using *EFP* and *DSBF*. Furthermore, by combining *EFP*, *DSBF*, and *BMF* together, the speed and energy efficiency are improved by 15.32× and 13.6× compared to the baseline. The accelerator takes only 0.53 ms and 0.13 mJ to process PointNeXt-S at 100 MHz with 0.9 V supply voltage.

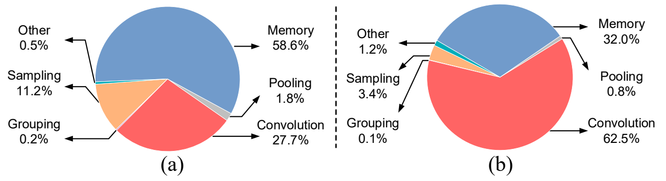

Fig. 7. (a) Area and (b) Energy breakdown of the proposed accelerator.

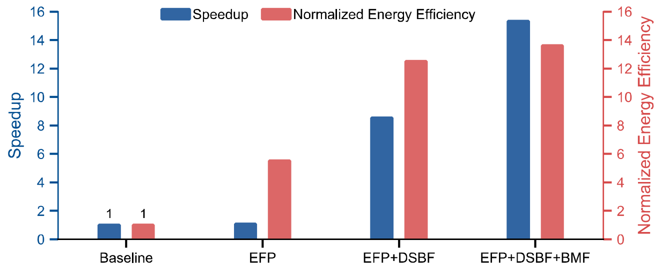

Fig. 8. The speedup and normalized effective energy efficiency by using the proposed techniques in PointNeXt-S.

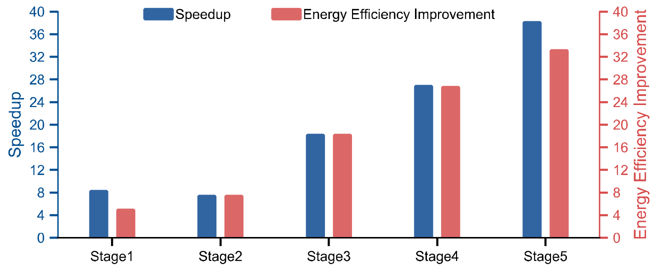

Fig. 9. The speedup and effective energy efficiency improvement of each stage in PointNeXt-S.

The speedup and effective energy efficiency of each stage in PointNeXt-S are demonstrated in Fig. 9. The speedup and energy efficiency exhibit significant variations across different stages. The latency of the first FPS in Stage1 dominates the overall latency of inference in the baseline, accounting for 93.4%. *DSBF* shortens the latency of Stage1 by 8.2×. Meanwhile, the latency and energy efficiency of all the stages are improved by *EFP*, especially in the deep stages, such as Stage4 and Stage5, where the channel sparsity of filters and features in these deep stages is largely improved, as mentioned in Section IV-A. Furthermore, *BMF* can eliminate the off-chip memory access of the large grouped intermediate features between MLPs in the shallow stages, as mentioned in Section IV-B. The application of *BMF* is further extended from shallow stages to deep stages of PNNs.

C. Comparisons with the State of the Art

The comparisons of the proposed point cloud accelerator with the state-of-the-art point cloud accelerators are detailed in Table I. As shown in Fig. 10, while running PointNeXt-S, the proposed accelerator improves the effective energy efficiency by 66.6×, 11.5×, 14.7×, 3.4×, 2.2×, 8.4×, and 3.1×, compared with Mesorasi [17], PointAcc [21], Point-X [16], Crescent [18], PNNPU [19], [14], and [15], respectively. The proposed accelerator reduces the per-frame energy consumption by 70.2×, 12.1×, 2×, 3.6×, 11.6×, 45.4×, and 15×, compared with Mesorasi [17], PointAcc [21], Point-X [16], Crescent [18], PNNPU [19], [14], and [15], respectively. Meanwhile, to evaluate the performance in terms of per-area and per-energy-consumption, an overall efficiency (Frame/mm^2/mJ) is compared. The proposed accelerator improves the overall efficiency by 74.9×, 32.3×, 9.1×, 3.9×, 126.8×, 127.5×, and 27×, compared with Mesorasi [17], PointAcc [21], Point-X [16], Crescent [18], PNNPU [19], [14], and [15], respectively.

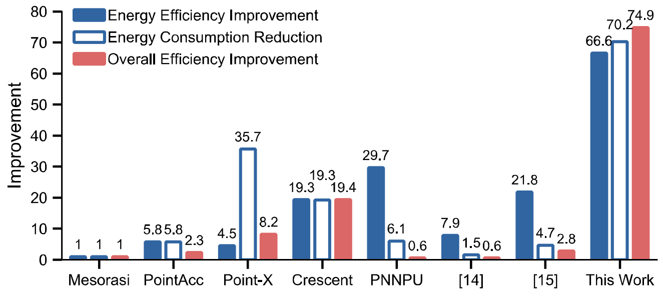

Fig. 10. The effective energy efficiency improvement, per-frame energy consumption reduction, and overall efficiency improvement, compared to the state-of-the-art works.

VI. Conclusions

In this paper, an energy-efficient real-time point cloud neural network accelerator is proposed. To reduce the high energy consumption and long latency caused by convolution, an efficient filter pruning is proposed. The redundant convolution of pruned filters and zero-value feature channels is skipped, while a large amount of inefficient off-chip memory access of grouping point features is eliminated. A block-wise multi-layer perceptrons (MLP) fusion is proposed to increase the on-chip reuse of features and significantly reduce off-chip memory access of point features in MLPs. Then, to eliminate the latency bottleneck of sampling, a dual-stream blocking method is proposed to enhance the parallelism of sampling while maintaining the inference accuracy. All the proposed techniques are enabled on the system architecture with specialized computation flow. The proposed point cloud neural network accelerator is implemented in the TSMC 28-nm CMOS technology. Benchmarked with the PointNeXt-S network classifying the ModelNet40 dataset, the proposed accelerator achieves an effective energy efficiency of 12.65 TOPS/W and a per-frame energy consumption of 0.13 mJ/frame, with all the optimization techniques while running at 100 MHz with 0.9 V supply voltage and 8-bit data width. Compared to the state-of-the-art point cloud neural network accelerators, the proposed accelerator enhances the energy efficiency and the per-frame energy consumption by up to 66.6× and 70.2×, respectively, thereby providing an attractive option for edge computing.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China 62074005 and Shenzhen Municipal Scientific Program JCYJ20200109140601691.

TABLE I
COMPARISONS WITH THE STATE OF THE ART

	Mesorasi [17] MICRO 2020	PointAcc [21] MICRO 2021	Point-X [16] MICRO 2021	Crescent [18] ISCA 2022	PNNPU [19] VLSI 2021	[14] VLSI 2022	[15] ISSCC 2023	**This work**
Technology	TSMC 16 nm	TSMC 40 nm	28 nm	TSMC 16 nm	65 nm	65 nm	28 nm	**TSMC 28 nm**
Area [mm^2]	1.55	3.9	6.8	1.55	16	4.1	2.69	**1.46**
SRAM [KB]	1624	274	545.4	1622.8	364	108.5	176	**128**
MAC	256	256	2048	256	4608	100	256	**1024**
Frequency [MHz]	1024	1024	1024	1024	50-200	50-300	40-450	**100**
Data Type	16b	A: 16b, W: 8b	A: 16b, W: 16b/8b	16b	8b	8b	8b	**8b**
Network Type	PointNet++	PointNet++	DGCNN, PointNet	PointNet++	VoteNet	MinkowskiNet	SECOND	**PointNeXt-S**
Effective Performance [GOPS]	467	1168.16	1888.92 @ DGCNN	1791.02 @ PointNet++	614.4 @ 200 MHz, 153.6 @ 50 MHz	62 @ 300 MHz, 10.3 @ 50 MHz	193.8 @ 400 MHz	**3003**
Power Consumption [mW]	2452.42	1057.08	2200 @ DGCNN	487.24 @ PointNet++	237 @ 200 MHz, 27.2 @ 50 MHz	80 @ 300 MHz, 6.9 @ 50 MHz	6.6 @ 60 MHz, 125 @ 400 MHz,	**237.4**
Effective Area Efficiency [GOPS/mm^2]	301.46	299.53	277.78 @ DGCNN	1155.5 @ PointNet++	0.55 @ 200 MHz, 0.14 @ 50 MHz	15.1 @ 300MHz, 2.5 @ 50 MHz	72 @ 400 MHz	**2056.9**
Effective Energy Efficiency [TOPS/W]	0.19	1.1	0.8586 @ DGCNN	3.68 @ PointNet++	2.59 @ 200 MHz, 5.65 @ 50 MHz	0.78 @ 300 MHz 1.5 @ 50 MHz	1.55 @ 400 MH, 4.14 @ 60 MHz	**12.65**
Energy Consumption [mJ/Frame]	8.92	1.54	1.6 @ DGCNN, 0.25 @ PointNet	0.46 @ PointNet++	3.2 @ 200 MHz, 1.47 @ 50 MHz	11.1 @ 300 MHz, 5.77 @ 50 MHz	5.0 @ 400 MHz, 1.9 @ 60 MHz	**0.13**
Frame Rate [Frame/s]	274.9	687.2	1307.1 @ DGCNN, 10000 @ PointNet	1053.4 @ PointNet++	84.4 @ 200 MHz, 21.1 @ 50 MHz	7.2 @ 300 MHz, 1.2 @ 50 MHz	16.9 @ 400 MHz, 3.3 @ 60 MHz	**1873**
Overall Efficiency [Frame/mm^2/mJ]	0.072	0.167	0.09 @ DGCNN, 0.59 @ PointNet	1.4 @ PointNet++	0.02 @ 200 MHz, 0.04 @ 50MHz	0.02 @ 300 MHz, 0.04 @ 50 MHz	0.07 @ 400 MHz, 0.20 @ 60 MHz	**5.39**

References

[1] B. Graham, M. Engelcke, and L. V. D. Maaten, "3D semantic segmentation with submanifold sparse convolutional networks," in *Proc. of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp. 9224-9232.

[2] C. Choy, J. Y. Gwak, and S. Savarese, "4D spatio-temporal convnets: Minkowski convolutional neural networks," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2019, pp. 3070-3079.

[3] J. Komorowski, "MinkLoc3D: Point cloud based large-scale place recognition," in *Proc. of the IEEE/CVF Winter Conference on Applications of Computer Vision*, Jan. 2021, pp. 1790-1799.

[4] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Transactions on Graphics*, vol. 38, no. 5, pp. 1-12, Oct. 2019.

[5] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2017, pp. 29-38.

[6] C. Wang, B. Samari, and K. Siddiqi, "Local spectral graph convolution for point set feature learning," in *Proc. of the Springer European Conference on Computer Vision*, Sept. 2018, pp. 56-71.

[7] J. Liu, B. Ni, C. Li, J. Yang, and Q. Tian, "Dynamic points agglomeration for hierarchical point sets learning," in *Proc, of the IEEE/CVF International Conference on Computer Vision*, Oct. 2019, pp. 7545-7554.

[8] Q. Xu, X. Sun, C.-Y. Wu, P. Wang, and U. Neumann, "Grid-GCN for fast and scalable point cloud learning," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2020, pp. 5661-5670

[9] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. the IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2017, pp. 77-85.

[10] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in *Proc. of the International Conference on Neural Information Processing Systems*, 2017, pp. 5105-5114.

[11] X. Ma, C. Qin, H. You, H. Ran, and Y. Fu, "Rethinking network design and local geometry in point cloud: A simple residual MLP framework," in *Proc. of the International Conference on Learning Representations*, Apr. 2022.

[12] G. Qian, Y. Li, H. Peng, J. Mai, H. A. A. K. Hammoud, M. Elhoseiny, and B. Ghanem, "PointNeXt: Revisiting PointNet++ with improved training and scaling strategies," in *Proc. of the International Conference on Neural Information Processing Systems*, Nov. 2022, pp. 23192-23204.

[13] H. Zhao, L. Jiang, C. W. Fu, and J. Jia, "Pointweb: Enhancing local neighborhood features for point cloud processing," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2019, pp. 5565-5573.

[14] Q. Cao and J. Gu, "A sparse convolution neural network accelerator for 3D/4D point-cloud image recognition on low power mobile device with hopping-index rule book for efficient coordinate management," in *IEEE Symposium on VLSI Technology and Circuits (VLSI) Dig. Tech. Papers*, Jun. 2022, pp. 106-107.

[15] W. Sun, X. Feng, C. Tang, S. Fan, Y. Yang, J. Yue, H. Yang, and Y. Liu, "A 28nm 2D/3D unified sparse convolution accelerator with block-wise neighbor searcher for large-scaled voxel-based point cloud network," in *IEEE International Solid-State Circuits Conference (ISSCC) Dig. Tech. Papers*, Feb. 2023, pp. 328-310.

[16] J.-F. Zhang and Z. Zhang, "Point-X: A spatial-locality-aware architecture for energy-efficient graph-based point-cloud deep learning," in *Proc. of the IEEE/ACM International Symposium on Microarchitecture*, Oct. 2021, pp. 1078-1090.

[17] Y. Feng, B. Tian, T. Xu, P. Whatmough, and Y. Zhu, "Mesorasi: Architecture support for point cloud analytics via delayed-aggregation," in *Proc. of the IEEE/ACM International Symposium on Microarchitecture*, Oct. 2020, pp. 1037-1050.

[18] Y. Feng, G. Hammonds, Y. Gan, and Y. Zhu, "Crescent: Taming memory irregularities for accelerating deep point cloud analytics," in *Proc. of the ACM/IEEE Annual International Symposium on Computer Architecture*, Jun. 2022, pp. 962-977.

[19] S. Kim, J. Lee, D. Im and H. -J. Yoo, "PNNPU: A 11.9 TOPS/W high-speed 3D point cloud-based neural network processor with block-based point processing for regular DRAM access," in *Proc. of Symposium on VLSI Circuits*, Jun. 2021, pp. 1-2.

[20] C. R. Qi, O. Litany, K. He, and L. J. Guibas, "Deep hough voting for 3d object detection in point clouds.," in *Proc, of the IEEE/CVF International Conference on Computer Vision*, Oct. 2019, pp. 9277-9286.

[21] Y. Lin, Z. Zhang, H. Tang, H. Wang, and S. Han, "PointAcc: Efficient point cloud accelerator," in *Proc. of the IEEE/ACM International Symposium on Microarchitecture*, Oct. 2021, pp. 449-461.

[22] Distiller. Accessed: Apr. 12, 2022. [Online]. Available: https://github.com/IntelLabs/distiller.

[23] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2015, pp. 1912-1920.

[24] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, S. Zheng, T. Lu, J. Gu, L. Liu, and S. Wei, "A high energy efficient reconfigurable hybrid neural network processor for deep learning applications," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 968-982, Apr. 2018.

[25] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proc. of the ACM/IEEE Annual International Symposium on Computer Architecture*, Jun. 2017, pp. 27-40.

[26] K. Simonyan, and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. of the International Conference on Learning Representations*, May 2015.