Adjustable Multi-Stream Block-Wise Farthest Point Sampling Acceleration in Point Cloud Analysis

Journal:	*IEEE Transactions on Circuits and Systems II: Express Briefs*
Manuscript ID	TCAS-II-19082-2023
Manuscript Type:	Regular Paper - Letters
Date Submitted by the Author:	14-Sep-2023
Complete List of Authors:	Jiao, Hailong; Peking University Shenzhen Graduate School Zhou, Changchun; Peking University Shenzhen Graduate School Fu, Yuzhe; Peking University Shenzhen Graduate School Ma, Yanzhe; Peking University Shenzhen Graduate School Han, Eryi; Reconova Technologies Co Ltd He, Yifan; Reconova Technologies Co Ltd
EDICS:	DCS110 - Digital ASICs < Digital Circuits and Systems (and VLSI), DCS160A5 - Low power architectures < DCS160 - Low power digital systems < Digital Circuits and Systems (and VLSI), DCS230 - Digital VLSI < Digital Circuits and Systems (and VLSI), DCS230B0 - VLSI digital circuits, designs and implementations < DCS230 - Digital VLSI < Digital Circuits and Systems (and VLSI)
TCAS-II Subject Category Please select the subject category that most closely fits with the scope of your manuscript:	Digital Circuits and Systems and VLSI

Adjustable Multi-Stream Block-Wise Farthest Point Sampling Acceleration in Point Cloud Analysis

Changchun Zhou*, Yuzhe Fu*, Yanzhe Ma, Eryi Han, Yifan He, *Member, IEEE,* and Hailong Jiao, *Member, IEEE*

***Abstract*—Point cloud is increasingly used in a variety of applications. Farthest Point Sampling (FPS) is typically employed for down-sampling to reduce the size of point cloud and enhance the representational capability by preserving contour points in point cloud analysis. However, due to low parallelism and high computational complexity, high energy consumption and long latency are caused, which becomes a bottleneck of hardware acceleration. In this brief, we propose an adjustable multi-stream block-wise FPS, adjusted by four configurable parameters, according to hardware and accuracy requirements. A unified hardware architecture is designed to implement the adjustable multi-stream block-wise FPS. Furthermore, we present a rapid searching algorithm to select the optimal configuration of the four parameters. Designed in an industrial 28-nm CMOS technology, the proposed hardware architecture achieves a latency of 0.005 ms and a frame energy consumption of 0.09 μJ/frame for 1 k input points at 200 MHz and 0.9 V supply voltage. Compared to the state of the art, the proposed hardware architecture reduces the latency by up to 84.38%, saves the energy by up to 76.19%, and improves the network accuracy by up to 1.05%.**

***Index Terms*—Point cloud neural networks, mapping, parallelism, FPS, acceleration framework, energy consumption, network accuracy, hardware architecture.**

I. Introduction

POINT cloud has recently gained popularity as a data source in a variety of fields, ranging from robotics [1], [2] and autonomous driving [3], [4] to augmented reality (AR) [5]. The advent of Light Detection and Ranging (LiDAR) technology has facilitated the rapid and precise collection of point cloud data, which contains valuable spatial representation information. However, the issues of disordered and discrete data in point cloud introduce considerable challenges for immediate applications. To tackle these issues, mapping operations (including sampling and grouping) are employed in point cloud neural networks (PNNs) [6]–[14], to build a mapping relationship between input point cloud and output point cloud of layers, enabling the hierarchical and strong feature extraction capability, thereby achieving high inference accuracy and robustness. Furthermore, sampling plays a crucial role in significantly reducing the input point cloud size and computational complexity of PNNs. Among various sampling methods, the farthest point sampling (FPS) algorithm has gained widespread adoption due to its ability to efficiently reduce the number of points while preserving shape and structural information. However, FPS is inherently sequential with a long latency of $O(N^2)$ cycles and a high computational complexity of $O(N^2)$, where N represents the number of points in the point cloud. Furthermore, FPS involves a substantial number of memory accesses of $O(N^2)$. Meanwhile, compared to a block of feature maps or a filter in convolutional neural networks, the grain of memory access in FPS is smaller with a size of $3 \times D$, where 3 and D are the number of dimensions and bit width of point coordinates, respectively. Therefore, FPS results in lower memory access efficiency. These make FPS a bottleneck in hardware acceleration for point cloud analysis. Therefore, an FPS acceleration framework that supports parallel computing with low computational complexity, low latency, and low memory access overhead, is desirable.

FPS acceleration has been investigated sporadically in recent years. In [15], a block-wise FPS algorithm is introduced to spatially divide point cloud into multiple blocks and perform local FPS inside each block. However, the block-wise FPS algorithm struggles to retain key details of point cloud when applied to smaller datasets, resulting in a large decrease in network accuracy. An adjustable FPS algorithm is introduced in [16]. Point cloud is segmented into multiple slices instead of blocks in [15], based on the spatial distribution. Similarly, the local FPS is then performed on each segment. However, if the data rearrangement mismatches the spatial distribution of point cloud, the network accuracy is significantly degraded. Furthermore, rearranging the point cloud data before processing leads to extra operations.

To solve these issues, an algorithm-hardware co-designed acceleration framework is proposed for FPS, aiming at optimizing network accuracy while minimizing hardware cost. An adjustable multi-stream block-wise FPS algorithm including four configurable parameters is proposed, which reduces the computational complexity and maintains the accuracy. A unified hardware architecture is developed to implement the proposed adjustable multi-stream block-wise FPS algorithm. A rapid searching algorithm is proposed to select the best software and hardware configurations for the proposed adjustable multi-stream block-wise FPS algorithm with specified priority between accuracy and energy consumption. The proposed FPS acceleration framework improves up to 1.05% network accuracy and reduces up to 76.19% energy consumption and 84.38% latency per frame,

This work was supported in part by the National Natural Science Foundation of China 62074005. (**Changchun Zhou and Yuzhe Fu contributed equally to this work.*) (*Corresponding author: Hailong Jiao.*)

Changchun Zhou, Yuzhe Fu, Yanzhe Ma, and Hailong Jiao are with the School of Electronic and Computer Engineering, Shenzhen Graduate School, Peking University, Shenzhen, 518055, China (e-mail: jiaohailong@pku.edu.cn). Eryi Han and Yifan He are with Reconova Technologies Co., Ltd., Xiamen, 361015, China.

compared to the state of the art. Designed in the TSMC 28-nm CMOS technology, the proposed hardware architecture achieves a latency of 0.005 ms and an energy consumption of 0.09 μJ/frame for 1 k input points at 200 MHz and 0.9 V supply voltage.

The paper is organized as follows. The original FPS algorithm and existing metrics to evaluate the similarity between two point clouds are introduced in Section II. The proposed adjustable multi-stream block-wise FPS acceleration framework is portraited in Section III. In Section IV, the experimental results are presented. The paper is concluded in Section V.

II. Background

A. The Original FPS Algorithm

FPS is typically employed in the sampling process of PNNs due to its ability to retain superior point cloud feature details, thereby enhancing network accuracy. In original FPS, the output points are iteratively sampled from the input point cloud in two distinct steps. As shown in Fig. 1, in the first step, FPS starts by randomly selecting an initial point ***a*** from the point cloud. The Euclidean distance between each unsampled point of {***b***, ***c***, ***d***, ***e***, ***f***} and the existing sampled point cloud {***a***} is computed and compared. Subsequently, point ***b*** with the maximum distance is added to the existing sampled point cloud {***a***} to form a new sampled point cloud {***a, b***}.

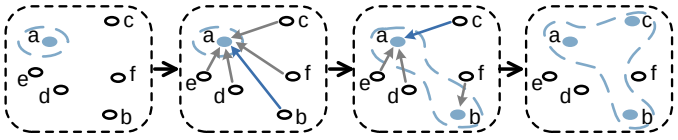

Fig. 1 The overall process of the original FPS.

In detail, the original FPS processing is inherently sequential with a long latency of $O(scale \times N)$ cycles and a high computational complexity of $O(scale \times N^2)$, where *scale* is determined by the sampling rate of FPS. Typically, in PNNs, the sampling rate is 1/2, establishing *scale* at 3/4.

B. Mahalanobis Distance and MAE

Mahalanobis distance (*MD*) serves as a metric to evaluate the similarity between a point and a point cloud. The formula to calculate *MD* between point *x* and point cloud *P* is presented as

$$D_M(x) = \sqrt{(x-u)^T S^{-1}(x-u)}, \qquad (1)$$

where *S* represents the covariance matrix of *P* and *u* signifies the mean vector across the dimensions of point cloud *P*. A smaller value of $D_M(x)$ indicates a less disparity between point *x* and point cloud *P*.

Mean Absolute Error (*MAE*) represents the average of the absolute differences between two point clouds. The formula of *MAE* is

$$MAE = \frac{1}{n}\sum_1^n |A_i - B_i|, \qquad (2)$$

where *n* is the total number of points in point clouds *A* and *B*. A smaller *MAE* indicates a higher similarity between the two point clouds.

III. Adjustable Multi-Stream Block-Wise FPS Acceleration Framework

A. Adjustable Multi-Stream Block-Wise FPS (AMB-FPS)

Due to the inherently sequential nature and high computational complexity of $O(N^2)$, FPS becomes a bottleneck in hardware acceleration for PNNs. To address these challenges, [15] introduces a block-wise FPS algorithm consisting of two principal steps: prediction and block-wise FPS. Two key parameters, namely the sparsity coefficient (*S*) and the number of blocks (*B*), are employed. An increase in the values of *S* and *B* tends to reduce the complexity and latency of FPS, yet at the cost of accuracy loss. In detail, a higher *S* leads to the dropout of critical contour points in the prediction step, thereby causing prediction distortion. Similarly, a larger *B* results in a smaller number of points in each block, which causes a substantial discrepancy between the predicted number and the actual number of points.

The adjustable multi-stream block-wise FPS algorithm (*AMB-FPS*) is proposed to further improve parallelism, reduce complexity, and maintain accuracy. *AMB-FPS* is composed of a multi-stream prediction stage and a multi-stream sampling stage, as shown in Fig. 2.

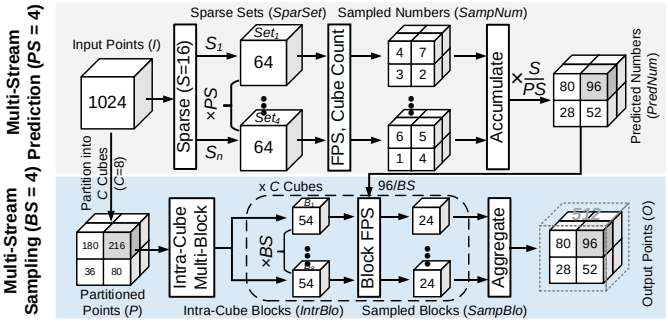

Fig. 2 The detailed mechanism of *AMB-FPS*.

AMB-FPS employs four key parameters. The sparsity coefficient (*S*), the number of prediction streams (*PS*), and the number of cubes (*C*) are used in the multi-stream prediction stage. *C* and the number of block streams (*BS*) are used in the multi-stream sampling stage. Algorithm 1 outlines the overall process of *AMB-FPS*. During the multi-stream prediction stage, *PS* distinct sparsity strategies are adopted to reduce the input points to *PS* small independent sparse point sets. FPS is performed on each set. Afterwards, the remained points of each set are partitioned into *C* cubes. The points are counted for each cube. The counted numbers in the same cube of *PS* sets are accumulated and then multiplied by *S/PS* to obtain the final predicted number of each cube in the original input point cloud. In the multi-stream sampling stage, the input point cloud is spatially partitioned into *C* cubes. Each cube is further uniformly sampled into *BS* blocks by adopting *BS* distinct

sparsity strategies. The predicted number of each cube obtained in the first stage is also divided by *BS*. FPS is performed on each block. The remained points of all blocks are aggregated to obtain the final sampled points. An example of *AMB-FPS* configured with *C=8*, *S=16*, and *PS=BS=4* is illustrated in Fig. 2. The number of points before and after FPS is 1024 and 512, respectively. In the prediction stage, the introduction of *PS* aims to mitigate the imprecision caused by the loss of key points due to a larger *S*. *BS* improves the computational parallelism and reduces the load imbalance across *C* cubes. Therefore, reasonable *PS* and *BS* are beneficial for improving accuracy and reducing latency, respectively. However, a large *PS* causes excessive computation during the prediction stage, while an exaggerated *BS* makes FPS resemble random sampling. To address these challenges, in Section III-C, a rapid searching algorithm is proposed to select a reasonable configuration (*C*, *S*, *PS*, *BS*) for *AMB-FPS*.

Algorithm 1 Overall Calculation for *AMB-FPS*

Input: Input Points I, Sampled Number N_O, C, S, PS, BS
Output: Output Points O
// Multi-Stream Prediction
$PredNum = []$
for ps in PS **do**
  $SparSet = Sparse(I, S_{ps})$
  $SampNum = CubeCount(FPS(SparSet, N_O/S))$
  $PredNum.append(SampNum)$
$PredNum.aggregate()$
// Multi-Stream Sample
$SampBlo = []$
$P = CubePartition(I, C)$
for c in C **do**
  $IntrBlo = BlockPartition(P[c], BS)$
  for bs in BS **do**
    $temp = FPS(IntrBlo[bs], PredNum[c]/BS)$
    $SampBlo[c].append(temp)$
  $SampBlo[c].aggregate()$
$O = SampBlo.toPointCloud()$
return O

B. Unified Hardware Architecture (UHA)

The unified hardware architecture of the proposed *AMB-FPS* acceleration framework is depicted in Fig. 3. *UHA* consists of a sampling module, a partition module, a memory module, a control module, and an interface module. The sampling module includes *M* cores, each of which can be configured to support different numbers of input points and output points. Each core performs the FPS of each block in Fig. 2. The partition module mainly consists of 16 comparators, counters, and buffers. The partition module is responsible for partitioning coordinates of point cloud into multiple blocks and performing sparsity strategies, providing inputs for the sampling module. The memory module is composed of *M*/2 static random-access memory (SRAM) banks, each bank housing *N*/*M*/2 (*N* is the number of input points) 256-bit words. An instruction set architecture (ISA) is designed for the off-chip host to control *UHA*. 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 sampling module, the partition module, and the allocation of the memory module. In the hardware architecture, the number of cores *M* is configured based on the number of blocks of *AMB-FPS* in Fig. 2. Then, the numbers of SRAM banks and SRAM words are configured to *M*/2 and *N*/*M*/2, respectively. Therefore, a hardware implementation that aligns with the proposed *AMB-FPS* is generated.

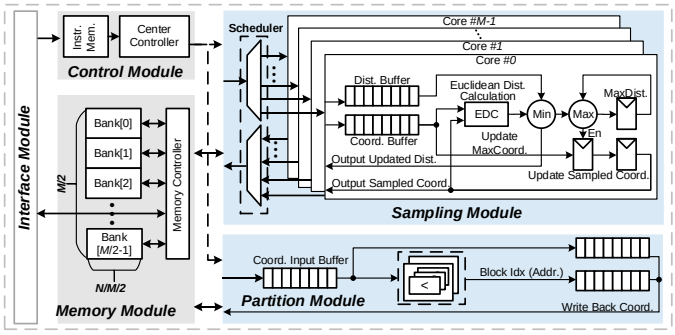

Fig. 3 The architecture of the proposed *UHA*.

C. Rapid Searching Algorithm (RSA)

All the configurations of *AMB-FPS* and *UHA* are designed for two goals: guaranteeing network accuracy and mitigating hardware energy consumption. However, to determine the optimal configuration, all the configurations need to be evaluated. For each configuration, replacing the original FPS by the configured *AMB-FPS*, retraining the network, and then evaluating the hardware are required. The entire process is significantly time-consuming and thus impractical. Therefore, a rapid searching algorithm (*RSA*) is proposed as shown in Algorithm 2, which rapidly generates the optimal configuration of *AMB-FPS* and hardware architecture for specified accuracy and hardware requirements.

Algorithm 2 Rapid Searching Algorithm

Input: w
Output: $[C, S, PS, BS, M]$
$C_l = [2, 4, 8, ...], S_l = [2, 4, 8, 16, ...]$
$PS_l = BS_l = [1, 2, 3...], M_l = [1, 2, 4, 8, 16, ...]$
$Config_l = []$
for C, S, PS, BS, M in $C_l, S_l, PS_l, BS_l, M_l$ **do**
  Compute *IMD*, *MAE*, *Acc*, *L*, *E*
  Compute $Loss(w)$
  $Config_l.append([C, S, PS, BS, M, Loss])$
$Config_l = Config_l.sort(Loss)$
$[C, S, PS, BS, M] = Config_l[0]$
return $[C, S, PS, BS, M]$

In Algorithm 2, *Loss* is introduced as a comprehensive evaluation index of *AMB-FPS* and *UHA*, which combines algorithm accuracy (*Acc*) and hardware energy consumption (*E*) as shown in (7). A lower *Loss* value indicates superior *AMB-FPS* performance. The weight w is manually controlled to prioritize either accuracy or hardware energy consumption for *Loss*. *Acc* is the sum of *MAE* and an improved Mahalanobis distance (*IMD*). *IMD* is explained in Section IV-A. *E* is comprised of hardware latency *(L)* and power consumption

(*Power*). L is the latency of the two stages in *AMB-FPS*. Therefore, L is affected by the configurations of the proposed *AMB-FPS* and hardware architecture. *Power* is derived from the hardware post-synthesis simulation by linear fitting.

$$L = ceil\left(\frac{PS}{Core}\right)\cdot\frac{3}{4}\left(\frac{N}{S}\right)^2 + ceil\left(\frac{C\times BS}{Core}\right)\cdot\frac{3}{4}\left(\frac{3N}{4C\times BS}\right)^2. \quad (3)$$

$$Power = 2.9 + 1.1\times M. \quad (4)$$

$$E = L\times Power. \quad (5)$$

$$Acc = IMD + MAE. \quad (6)$$

$$Loss = w\times Acc + (1-w)\times E. \quad (7)$$

In Algorithm 2, for each configuration (C, S, PS, BS, M), *Acc* and *E* are computed. *Loss* for each configuration is calculated according to (7) and is added to $Config_l$. Subsequently, the configuration in $Config_l$ with the smallest *Loss* is selected as the best configuration for *AMB-FPS* and hardware architecture.

IV. Experimental Results

A. Analysis of Software and Hardware Implementations

To evaluate the accuracy of the proposed *AMB-FPS*, the PointNeXt-S network classifying ModelNet40 [17] dataset is trained in PyTorch as the benchmark. The open-source tool Distiller [18] is used for pruning, quantization to 8-bit data width, and retraining PointNeXt to recover the accuracy. *UHA* is designed in the TSMC 28-nm high-performance computing CMOS technology. Cadence Genus is used for logic synthesis at 200 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 original *MD* only represents the distance between two points or between a point and a point cloud. Therefore, *IMD* is proposed to reflect the similarity between two point clouds. The mean value of each dimension for the two point clouds (P_1 and P_2) is computed to derive two representative center points, which are u_1 and u_2. Subsequently, the covariance matrices of the two point clouds, namely S_1 and S_2, are calculated and accumulated to obtain an overall covariance matrix. The *IMD* between the point clouds is then computed based on

$$IMD(P_1,P_2) = \sqrt{(u_1-u_2)^T(S_1+S_2)^{-1}(u_1-u_2)}. \quad (8)$$

An *IMD* value of 0 signifies that the input point clouds are identical. Conversely, a larger *IMD* value indicates a larger difference between the two point clouds. The impact of diverse configurations of *AMB-FPS* on *Acc* and *E* are depicted in Fig. 4. The output point cloud of the original FPS is employed as a baseline for computing *Acc*. A larger *C* causes a larger error in the prediction stage due to the linear decrease in the number of points in each cube. Consequently, *Acc* becomes worse, as shown in Fig. 4a. However, the reduction in points per cube induces a quadratic complexity reduction in the FPS of each cube, thus leading to a decline in *E*. As *S* increases, the sparse step is more aggressive in the prediction stage, which leads to the deterioration of *Acc*. In terms of *E*, an increase in *S* leads to fewer points performed by FPS for each set during the prediction stage in Fig. 2, thus contributing to a reduced prediction latency and resulting in a smaller *E*.

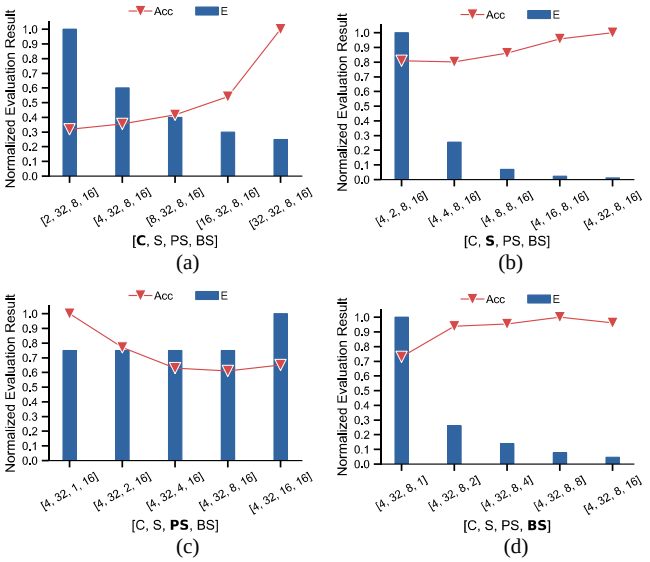

Fig. 4. The impact of different configurations of *AMB-FPS* on *Acc* and *E*. (a) The numbers of cubes (*C*). (b) The sparse coefficient (*S*). (c) The numbers of prediction streams (*PS*). (d) The number of block streams (*BS*).

The parameter *PS* serves as a multi-stream prediction mechanism designed to compensate for the accuracy loss caused by aggressive *S*. However, a large *PS* causes excessive computation and worse *E* during the prediction stage without improving *Acc* in (6), as $PS \geq 8$ (M=8) in this case. *BS* is used in the multi-stream sampling stage, which reduces hardware complexity by reducing the number of points in each block. Therefore, as shown in Fig. 4d, a larger *BS* leads to a smaller *E*. However, *BS* cannot be set to be excessively large, which reduces the number of points per block during the sampling stage and makes FPS resemble random sampling. Meanwhile, if only *BS* is employed (such as [*C*, *S*, *PS*, *BS*] = [1, 0, 0, 16] for *AMB-FPS*), the global features of point cloud are lost after uniformly sampling into *BS* blocks, leading to a significant network accuracy loss (>2%). Therefore, the introduction of other parameters [*C*, *S*, *PS*] is necessary to alleviate the impact of *BS*. The partition of input point cloud into *C* cubes preserves the global features. Meanwhile, *S* and *PS* enable the accurate prediction of the sampled number in each cube.

B. Comparisons with the State of the Art

To verify the effectiveness of the proposed adjustable multi-stream block-wise FPS acceleration framework, we perform a comparative analysis with the original FPS and PNNPU [15]. Furthermore, for a comprehensive algorithm comparison, algorithm accuracy indicators, such as *MAE*, *IMD*, and network overall accuracy (OA), and hardware comparative indicators, such as *L* and *E*, are employed. To further compare the original FPS, PNNPU [15], and the proposed *AMB-FPS*, the original FPS in PointNeXt is replaced by the block-wise FPS algorithm

of PNNPU and the proposed *AMB-FPS*. Then, the network is retrained to recover the accuracy. The results are presented in Table I.

TABLE I
PERFORMANCE COMPARISONS OF DIFFERENT METHODS

	Original FPS	PNNPU [15] VLSI 2021	AMB-FPS-v0	AMB-FPS-v1
[*C*, *S*, *PS*, *BS*]	N/A	[16, 16, 1, 1]	[4, 32, 2, 16]	[2, 32, 16, 15]
Core (*M*)	1	16	64	30
w : (1-*w*) (/100000)	N/A	N/A	1:10	10:1
MAE	0	1.237	0.030	0.011
IMD	0	0.169	0.128	0.054
Network OA	91.45	90.36	91.09	91.41
L (*ms*)	3.932	0.031*	0.005	0.008
E (*μJ*)	15.729	0.376*	0.090	0.236

*The simulation results of the reimplemented FPS hardware.

In Table I, AMB-FPS-v0 represents the hardware-prioritized version, which mainly optimizes *L* and *E* with less attention on *MAE* and *IMD*. AMB-FPS-v1 signifies the accuracy-prioritized version, focusing on improvements of *MAE* and *IMD* yet with hardware performance loss. Compared to the original FPS, our AMB-FPS-v0 (AMB-FPS-v1) reduces 99.87% (99.8%) and 99.43% (98.5%) in *L* and *E*, respectively, while ensuring negligible network accuracy loss (<0.4%). Compared with PNNPU, the AMB-FPS-v0 (AMB-FPS-v1) reduces 97.56% (99.11%), 23.08% (67.74%), 84.38% (74.36%), and 76.19% (37.5%) in *MAE*, *IMD*, *L*, and *E*, respectively. In terms of network accuracy, AMB-FPS-v0 and AMB-FPS-v1 surpass PNNPU by 0.73% and 1.05%, respectively.

V. CONCLUSIONS

In this brief, an algorithm-hardware co-designed FPS acceleration framework is proposed to reduce energy consumption while maintaining accuracy. To improve parallelism and reduce energy consumption, an adjustable multi-stream block-wise FPS algorithm is proposed. A unified hardware architecture for the adjustable multi-stream block-wise FPS algorithm is designed. A rapid searching algorithm is developed to select the optimal configuration of the adjustable multi-stream block-wise FPS algorithm and the unified hardware architecture under specified priority between accuracy and energy savings. Benchmarked with PointNeXt-S network classifying ModelNet40 and designed in the TSMC 28-nm CMOS technology, the proposed FPS acceleration framework achieves a latency of 0.005 ms and a frame energy consumption of 0.09 μJ/frame for 1 k input points at 200 MHz and 0.9 V supply voltage. Compared to the state of the art, the proposed FPS acceleration framework reduces up to 84.38% latency, saves up to 76.19% energy consumption, and improves up to 1.05% network accuracy.

REFERENCES

[1] M. Wen, Y. Dai, T. Chen, C. Zhao, J. Zhang, and D. Wang, "A robust sidewalk navigation method for mobile robots based on sparse semantic point cloud," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2022, pp. 7841-7846.
[2] Y. Cheng, J. Su, M. Jiang, and Y. Liu, "A novel radar point cloud generation method for robot environment perception," *IEEE Trans. Robot.*, vol. 38, no. 6, pp. 3754-3773, Dec. 2022.
[3] Y. Li *et al.*, "Deep learning for LiDAR point clouds in autonomous driving: a review," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 8, pp. 3412-3432, Aug. 2021.
[4] R. Abbasi, A. K. Bashir, H. J. Alyamani, F. Amin, J. Doh, and J. Chen, "Lidar point cloud compression, processing and learning for autonomous driving," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 1, pp. 962-979, Jan. 2023.
[5] L. Kästner, V. C. Frasineanu, and J. Lambrecht, "A 3d-deep-learning-based augmented reality calibration method for robotic environments using depth sensor data," in *Proc. Int. Conf. Robot. Automat. (ICRA)*, 2020, pp. 1135-1141.
[6] 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. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2017, pp. 5105-5114.
[7] 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. Int. Conf. Learn. Represent. (ICLR)*, Apr. 2022.
[8] G. Qian *et al.*, "PointNeXt: Revisiting PointNet++ with improved training and scaling strategies," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, Nov. 2022, pp. 23192-23204.
[9] H. Zhao, L. Jiang, C. W. Fu, and J. Jia, "Pointweb: Enhancing local neighborhood features for point cloud processing," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 5565-5573.
[10] L. Li, L. He, J. Gao, and X. Han, "PSNet: Fast data structuring for hierarchical deep learning on point cloud," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 10, pp. 6835-6849, Oct. 2022.
[11] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "Pointcnn: Convolution on x-transformed points," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 31, 2018.
[12] W. Wu, Z. Qi, and L. Fuxin, "Pointconv: Deep convolutional networks on 3d point clouds," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 9621–9630.
[13] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, "Point transformer," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, 2021, pp. 16 259–16 268.
[14] Y. Zhang, Q. Hu, G. Xu, Y. Ma, J. Wan, and Y. Guo, "Not all points are equal: Learning highly efficient point-based detectors for 3d lidar point clouds," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2022, pp. 18 953–18 962.
[15] 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. Symp. VLSI Circuits*, Jun. 2021, pp. 1-2.
[16] J. Li, J. Zhou, Y. Xiong, X. Chen, and C. Chakrabarti, "An adjustable farthest point sampling method for approximately-sorted point cloud data," in *Proc. IEEE Int. Workshop Signal Process. Syst. (SiPS)*, Oct. 2022, pp. 1-6.
[17] Z. Wu *et al.*, "3d shapenets: A deep representation for volumetric shapes," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1912-1920.
[18] Distiller. Accessed: Sep. 14, 2023. [Online]. Available: https://github.com/IntelLabs/distiller.