

# Parallel-SA: Point Cloud Processing Acceleration via Parallel Set Abstraction

Dongdong Tang<sup>1,3</sup>, Weilan Wang<sup>1,3\*</sup>, Yu Mao<sup>1</sup>, Wenjing Xie<sup>1</sup>, Nan Guan<sup>1</sup>, Tei-Wei Kuo<sup>2</sup>, and Chun Jason Xue<sup>3</sup>

<sup>1</sup>City University of Hong Kong, <sup>2</sup>National Taiwan University, <sup>3</sup>MBZUAI

**Abstract**—Point-based networks achieve high accuracy by preserving the intrinsic spatial structure of point clouds. The spatial information is effectively extracted by set abstraction, a critical module for feature learning in point-based networks. However, set abstraction introduces a computational bottleneck, and naive parallelization often degrades sampling quality, leading to accuracy loss. To address these challenges, we propose Parallel-SA, a framework that accelerates point-based networks by transforming set abstraction from sequential to parallel processing without sacrificing accuracy. Parallel-SA leverages a multi-scale sampling distribution approximation to preserve sampling quality under parallel execution. In addition, it employs distribution-aware balanced partitioning and adaptive load-balancing refinement to further improve efficiency. Experiments show that Parallel-SA achieves an average  $2.38\times$  speedup in set abstraction with minimal accuracy degradation.

## I. INTRODUCTION

Point cloud is a 3D collection of points that captures rich geometric and spatial information for accurate perception of complex environments [1]–[3], [24]. 3D object detection is particularly critical as it allows the system to recognize and locate individual objects. Point-based networks achieve state-of-the-art accuracy by detecting objects directly from raw points, in contrast to other methods that convert point clouds into grids or images. This direct processing preserves fine-grained patterns and local structures, but comes at a substantial computational cost. To address this limitation, we propose Parallel-SA, a framework that improves the efficiency of point-based networks without sacrificing accuracy.

Point-based networks suffer from significant processing latency, primarily due to the set abstraction module, which is the core feature extraction component in most state-of-the-art architectures [30], [33], [34]. Set abstraction selects a spatially uniform subset of points for feature extraction, which is crucial for handling the non-uniform density of point clouds. It preserves fine details in sparse regions while preventing over-sampling in dense regions. Set abstraction achieves this by adaptively fusing multi-scale features and weighting them based on local density [4], enabling effective representation learning and accurate 3D detection. Despite these strengths, its sequential processing and high computational complexity create a performance bottleneck, accounting for 40%–90% of the total detection latency in point-based networks.

This paper proposes Parallel-SA, a framework for efficient parallel execution of set abstraction while preserving the accuracy of point-based detection networks. The main challenge of

parallelizing set abstraction lies in its first sampling stage. Set abstraction captures point cloud structures through three stages: sampling, grouping, and PointNet. In the sampling stage, farthest point sampling (FPS) is used to select representative points for feature extraction. FPS produces spatially uniform samples, where selected points are evenly distributed across the geometric space. It achieves this by iteratively selecting the point farthest from all previously chosen points. This spatially uniform sampling is fundamental to set abstraction as it enables robust feature learning across diverse sampling conditions. However, parallelizing FPS is non-trivial. A naive strategy that partitions the raw point cloud and applies FPS independently to each partition cannot maintain spatial uniformity at the global level. Although FPS achieves uniform sampling within each local partition, this approach lacks the global perspective needed to ensure uniform sampling across the entire point cloud space. The central challenge is determining an appropriate number of samples for each partition so that the combined result approximates the outcome of global FPS achieved through direct sampling of the whole point cloud. This allocation cannot be directly known in advance, since the target distribution of samples is only revealed after performing FPS sequentially on the entire point cloud.

Parallel-SA introduces several key innovations to overcome parallelization challenges. Parallel-SA transforms set abstraction from an inherently sequential process into parallel processing, maintaining accuracy and significantly improving computational efficiency through two key contributions: (1) We propose a multi-scale sampling distribution approximation that efficiently estimates the optimal number of samples for each partition. This enables parallel FPS to approximate the outcome of global FPS across the entire point cloud, thereby preserving sampling quality and maintaining detection accuracy. (2) To achieve higher efficiency in parallel set abstraction, workload imbalance across partitions must be minimized. Otherwise, partitions with large computational workloads will create processing bottlenecks that degrade overall system performance. Our design maximizes parallel efficiency by implementing workload balancing to minimize worst-case latency. We introduce distribution-aware balanced partitioning and adaptive load balance refinement to ensure balanced computational distribution across partitions. Based on experimental results, Parallel-SA can achieve  $2.38\times$  speedup on average in set abstraction with only minimal accuracy loss. In general, this paper makes the following contributions:

- We present a multi-scale sampling distribution approxima-

\*Corresponding author

tion method that ensures parallel FPS closely approximate the output of global FPS on the entire point cloud, preserving the quality of spatially uniform sampling.

- We propose distribution-aware balanced partitioning to balance workloads and prevent long-latency partitions.
- To further enhance the overall efficiency, we introduce adaptive load balance refinement to adjust the workload of partitions with long processing latency.
- We present extensive experiments to evaluate the performance of Parallel-SA, and Experimental results show that Parallel-SA achieves an average speedup of 2.38 $\times$  in set abstraction with minimal accuracy loss.

## II. BACKGROUND AND RELATED WORK

### A. Point-based Object Detection

A point cloud is a set of 3D data points  $(x, y, z)$  providing spatial and geometric information. Object detection is a perception task that enables reliable and accurate estimation of the surrounding environment [1], [25]. PointNet++ [4] proposes the *set abstraction* module, which is widely adopted in state-of-the-art point-based networks [9], [33], [34] for feature extraction. It performs hierarchical feature learning through three sequential stages:

**Sampling:** Set abstraction employs farthest point sampling (FPS) to iteratively select a subset of points  $S_p$  from the input point set  $S=\{x_1, x_2, \dots, x_n\}$ . FPS begins with a random point from  $S$ , then iteratively selects the point  $x_i$  that has the maximum distance to the current  $S_p$ . The distance of  $x_i$  to  $S_p$  is:  $Distance(x_i, S_p) = \min_{x_j \in S_p} (Distance(x_i, x_j))$ . FPS achieves spatially uniform sampling, with points evenly spread across the point cloud’s geometric space, ensuring comprehensive structural representation for robust feature learning.

**Grouping:** This stage constructs local region sets by finding neighboring points around each sampled point in  $S_p$  using either ball query (up to  $K$  neighbors within a radius) or kNN (the  $K$  closest points).

**PointNet:** The local regions for each sampled point in  $S_p$  are encoded into feature vectors and PointNet is applied [5] to extracts point-to-point relations in the local region.

### B. Related Work

Several works have been proposed to improve the performance of point cloud processing in recent years [10], [14]–[22], [26], [27]. Mesorasi [6] proposes an algorithm-architecture co-design that enables parallel execution of neighbor search and feature computation, thereby enhancing the system’s efficiency. PointAcc [8] presents a versatile design to support diverse mapping operations that find the nonzero neighbors and the nonzero output point cloud corresponding to each weight. FusionArch [37] identifies the serial execution of sampling, redundant feature computation, and redundant memory accesses as a system bottleneck. It designs an ASIC-based accelerator with software-hardware co-design to implement optimizations in hardware. Some studies target the optimization of sampling in point cloud processing [28], [36]. QuickFPS [28] proposes a hardware acceleration design for Farthest Point Sampling (FPS)

in large-scale point clouds. It presents a bucket-based approach with a two-level tree structure, which intelligently reduces the memory and compute costs by processing only essential spatial buckets.

## III. MOTIVATION

To investigate the bottlenecks in point-based networks, we perform a latency analysis conducted on NVIDIA GTX 2080Ti. As illustrated in Figure 1, set abstraction creates a significant performance bottleneck, accounting for 42.5%, 92.1% and 88.2% of the total processing latency per frame in PointRCNN [30], IASSD [34] and 3DSSD [33], respectively. Figure 2 presents the detailed latency breakdown of set abstraction in 3DSSD [33]. The set abstraction module processes point clouds hierarchically in three levels, each applying sampling, grouping, and PointNet stages to extract features across multiple scales. Level 1 dominates the total processing latency, and Levels 2,3 demonstrate substantially reduced latency due to their processing of fewer points. To address this bottleneck, we propose Parallel-SA, which introduces fine-grained parallelism by partitioning the computation within each level, enabling more substantial speedups.

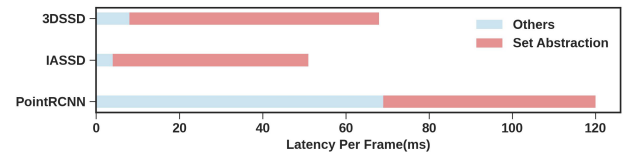


Fig. 1: Latency analysis evaluated on PointRCNN [30], 3DSSD [33] and IASSD [34].

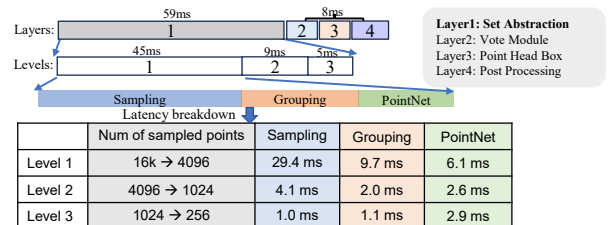


Fig. 2: Detailed latency breakdown of set abstraction. Each level includes sampling, grouping and PointNet stages.

## IV. DESIGN CHALLENGES

Parallelizing set abstraction is challenging due to the inherent properties of point cloud data: its unordered structure and non-uniform density across spatial regions. These characteristics present two challenges for the parallel implementation of the set abstraction.

**Challenge 1: Maintaining spatially uniform sampling in parallel set abstraction.** FPS achieves spatially uniform sampling results, with sampled points evenly distributed across the point cloud’s geometric space, as demonstrated in Figure 3. This approach prevents oversampling in dense regions while preserving critical details in sparse areas, achieving a more comprehensive feature representation. Naive parallel

approaches split the  $N$  points into  $n$  partitions and sample  $K/n$  points from each, but this strategy cannot maintain spatial uniformity across the whole point cloud. While the sampled points are uniformly distributed within each partition, the overall sampling results lack spatial uniformity across the entire space. Specifically, sparse regions suffer from undersampling and poor feature extraction, leading to accuracy loss. Random sampling is another downsampling method used in point cloud learning [13]. Despite its high efficiency, random sampling cannot achieve spatial uniformity and tends to oversample or undersample certain areas. As shown in Figure 3, it follows the raw point cloud’s density pattern, with more samples in dense regions and fewer in sparse regions.

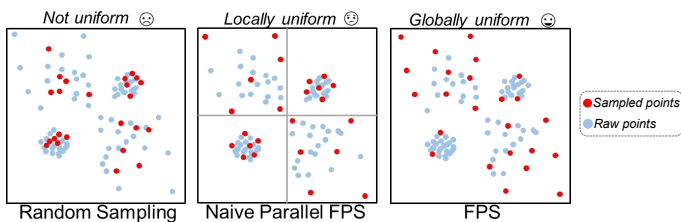


Fig. 3: Point distribution of random sampling, naive parallel FPS, and the original FPS.

**Challenge 2: Balancing workloads across partitions to maximize processing efficiency.** The time complexity of sampling  $K$  points from  $N$  raw points is  $O(KN)$ , while grouping scales with  $K$ . In parallel set abstraction, processing time is dominated by the slowest partition, so minimizing worst-case latency is essential. An effective parallel FPS algorithm requires a balanced workload across all partitions, which in turn demands equalizing both raw points  $N$  and sampled points  $K$  across partitions. Designing an effective partitioning strategy for parallel FPS presents challenges, as it must carefully account for the distribution of raw and sampled points.

## V. METHODOLOGY

### A. Overview

Figure 4 provides an overview of Parallel-SA. Firstly, the multi-scale sampling distribution approximation estimates the

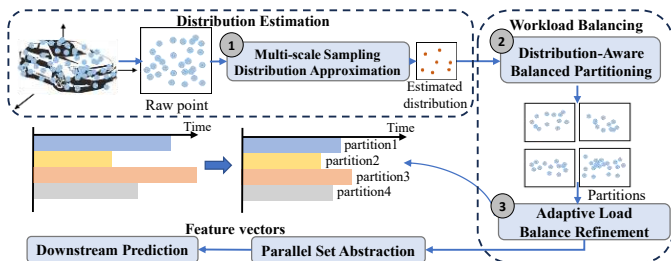


Fig. 4: System overview of Parallel-SA. Multi-scale sampling distribution approximation (section V-B) estimates the distribution of uniformly sampled points. Distribution-aware balanced partitioning (section V-C) and adaptive load balance refinement (section V-D) further enhance the overall efficiency.

distribution of spatially uniform sampled points. This estimation ensures global spatial uniformity in sampling, so the results of parallel FPS closely match those from FPS on the entire point cloud. Based on the input raw points and estimated distribution, distribution-aware balanced partitioning defines the optimal partitioning that balances computational workloads across partitions. The optimal sampling number in each partition is determined based on the estimated distribution of sampled points, followed by adaptive load balance refinement to maximize computational efficiency.

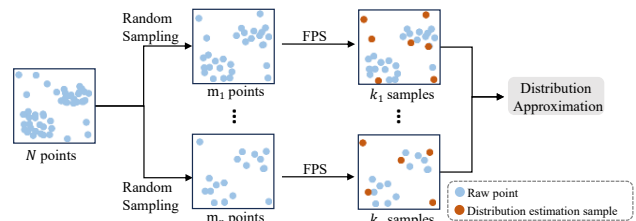


Fig. 5: Workflow of Multi-scale Sampling Distribution Approximation.

### B. Multi-scale Sampling Distribution Approximation

Maintaining global spatial uniformity of sampling results in parallel set abstraction requires predetermining the optimal sampling number for each partition. When each partition samples its optimal sampling number, parallel FPS results effectively approximate those from FPS on the entire point cloud. To determine the optimal sampling number, we propose a multi-scale sampling distribution approximation to efficiently estimate the distribution of points sampled uniformly across point cloud space.

Our approach stems from the observation that random sampling preserves point distribution, selecting more points in dense areas and fewer in sparse regions. Consequently, the downsized raw points maintain the distribution pattern of the input point cloud. When FPS is applied to downsized raw points, it generates samples by iteratively selecting points farthest from previously selected points. Therefore, we can infer that points sampled from downsized raw data maintain a distribution similar to those obtained by sampling the entire point cloud. FPS has a time complexity of  $O(KN)$  for obtaining  $K$  sampled points from  $N$  raw points. Sampling on downsized raw points achieves substantially lower latency than sampling the complete point cloud, while maintaining distribution properties that facilitate determining the optimal sampling number for each partition.

Figure 5 demonstrates the multi-scale sampling distribution approximation for an input point cloud of  $N$  raw points. The input is first downsized through a multi-scale random sampling to obtain  $m_1, m_2, \dots, m_n$  points. FPS samples  $k_1, k_2, \dots, k_n$  points respectively from these downsized sets with a consistent sampling rate of 25% across all scales. These points serve as distribution estimation samples for approximating the distribution, not actual sampling results for subsequent grouping and PointNet stages. The distribution estimation samples from

each scale are aggregated to approximate the distribution of points that would be obtained by sampling the original input. The multi-scale approach reduces estimation variance and is less sensitive to random initialization compared to single-scale methods, providing a more stable distribution approximation.

### C. Distribution-Aware Balanced Partitioning

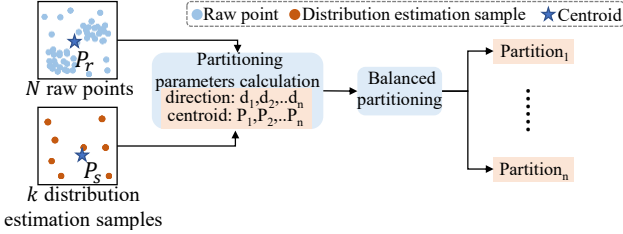


Fig. 6: Distribution-Aware Balanced Partitioning for Set Abstraction. The partitioning considers both the distribution of  $N$  raw points and  $k$  distribution estimation samples.

In parallel processing, balanced computational workloads across partitions are essential for maximizing efficiency, as partitions with larger workloads create bottlenecks that degrade overall performance. We propose distribution-aware balanced partitioning that leverages the distribution of both raw points and distribution estimation samples to achieve a balanced computational workload across partitions.

In set abstraction, sampling  $K$  points from  $N$  requires  $O(KN)$  time, and grouping latency scales with  $K$ . Thus, balancing workloads requires equalizing raw points and sampling across partitions. Based on this insight, we determine the partitioning of parallel set abstraction using the distribution of raw points and distribution estimation samples. Figure 6 illustrates the process of partitioning set abstraction where  $K$  points are sampled from  $N$  raw points. The partitioning is guided by  $k = \sum_{i=1}^n k_i$  distribution estimation samples obtained from multi-scale sampling, where  $k_i$  estimation samples are obtained from  $m_i$  downsized raw points at each scale.

To determine the optimal partitioning, we first identify the splitting direction by maximizing the combined variance of the raw point set  $R$  and the distribution estimation samples  $S$ :  $d_i = \arg \max_{d \in x, y} (\sigma_R^2(d) + \sigma^2 S(d))$ , where  $\sigma_R^2(d)$  and  $\sigma_S^2(d)$  represent the coordinate variances of the raw point set  $R$  and the distribution estimation samples  $S$  along direction  $d$ . This variance-based approach ensures we partition along the axis exhibiting the greatest spatial dispersion in both point distributions

After identifying the optimal splitting direction, we compute the median coordinates for both the raw point set  $R$  and distribution estimation samples  $S$ , denoted as  $P_r(x_r, y_r, z_r)$  and  $P_s(x_s, y_s, z_s)$ , respectively. The final partitioning point  $P_b$  is derived through a weighted combination of these medians  $P_b = \alpha \cdot P_r + (1 - \alpha) \cdot P_s$ , where  $\alpha$  represents the weighting coefficient that balances the contribution between the median of raw points  $P_r$  and the median of samples  $P_s$ . The weighting coefficient  $\alpha$  is computed based on the size of the point sets. Let

$N_r$  and  $N_s$  denote the size of point sets  $R$  and  $S$ , respectively. The weight  $\alpha$  is calculated as  $\alpha = \frac{1}{1 + e^{-r}}$ , where  $r = \frac{N_r}{N_r + N_s}$ .

The weighting coefficient  $\alpha$  is determined through a two-stage process. First, we compute the size ratio  $r$ , which assigns greater weight to larger point clouds to minimize overall partitioning imbalance. The imbalance for point partitioning is defined as the absolute difference in the number of points between two halves. While this weighting may lead to relatively higher imbalance for smaller point clouds, their absolute imbalance remains within acceptable bounds. Second, we apply a sigmoid transformation to  $r$  to create a nonlinear mapping. This transformation prevents marginalization of small point clouds ( $N_r \gg N_s$ ), ensuring distribution estimation samples retain non-negligible weight.

With the partitioning point  $P_b$  and direction  $d$ , we apply the balanced partitioning for both raw points and distribution estimation samples. The splitting produces approximately balanced partitions for both point sets. This approach can be applied recursively for partitioning and subsequent partitioning parameters are determined within each partition to further split the points. As a result, raw points are divided into  $n$  partitions with  $N = \sum_{i=1}^n N_i$ , where  $N_i$  is the number of raw points in partition  $i$  and distribution estimation samples are divided into  $n$  partitions with  $k = \sum_{i=1}^n k'_i$ , where  $k'_i$  is the number of samples in partition  $i$ .

Given the partitions, we estimate the optimal sampling number for each partition to allocate the total  $K$  samples into  $n$  parts:  $K = \sum_{i=1}^n K_i$ . Since the distribution estimation samples share a similar distribution with samples from the entire point cloud, we can infer that  $\frac{k'_i}{k} \approx \frac{K_i}{K}$ . Based on this, we estimate the optimal sampling number as  $K_i = K \frac{k'_i}{k}$ . As a result, the distribution-aware balanced partitioning divides the set abstraction into  $n$  partitions, where partition  $i$  contains  $N_i$  raw points and samples  $K_i$  points according to our estimation. This parallel set abstraction strategy effectively approximates the set abstraction on the entire point cloud.

### D. Adaptive Load Balance Refinement

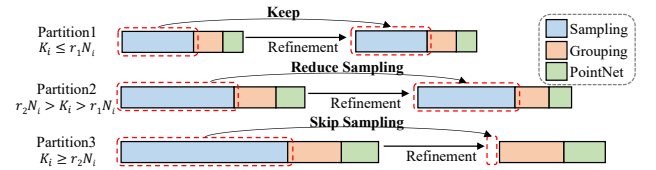


Fig. 7: Adaptive load balance refinement adjusts each partition's load based on  $K_i, N_i$ .

Distribution-aware balanced partitioning divides the input into  $n$  approximately even partitions:  $N = \sum_{i=1}^n N_i$ ,  $K = \sum_{i=1}^n K_i$ . However, the resulting workload distribution is not even. Some partitions contain more points, leading to longer latency and bottlenecks in parallel processing. Since overall latency is bound by the slowest partition, performance optimization requires minimizing this worst-case latency. To further improve the efficiency of parallel processing, we propose

adaptive load balance refinement to address these partitions with higher latency.

Figure 7 illustrates the partitions in parallel set abstraction under varying  $K_i$  conditions. The processing latency for sampling, grouping, and PointNet stages exhibits a proportional relationship with  $K_i$  values. The key principle of adaptive load balance refinement is based on the fact that FPS has an optimal sampling rate  $r$ , which typically approximates 25% in point-based networks [30], [33], [34]. At this sampling rate, set abstraction effectively extracts features from raw point clouds to make reliable detection. Sampling beyond this optimal rate ( $K > rN$ ) leads to only marginal accuracy improvements, as additional sampled points contribute minimally to enhanced feature learning or detection capability. The extra sampling introduces unnecessary computational overhead, since the latency of FPS is proportional to the number of sampled points. To achieve refinement, we define two sampling rate thresholds  $r_1$  and  $r_2$  ( $r_2 > r_1$ ). Our adaptive load balance refinement strategy is implemented in three cases:

- For partition  $i$ , when  $K_i \leq r_1 N_i$ , the sampling in this partition remains unchanged because the partition with small sample sizes has relatively low latency. These partitions do not create system bottlenecks.
- When  $r_1 N_i < K_i < r_2 N_i$ , sampling is set to  $K_i = r_1 N_i$ .  $r_1$  is set to 25% that balances the accuracy and efficiency. In these cases, set abstraction obtains a sufficient number of sampled points to extract discriminative features from raw point clouds for reliable detection. Additional sampled points contribute minimally to enhanced detection capability while introducing unnecessary computational latency.
- $K_i > r_2 N_i$  indicates a high sampling rate, where most raw points in the partition would be selected. In such cases, we skip the sampling process and select all points. Though this increases the latency of subsequent grouping and PointNet stages due to the increased number of sampled points, the overall processing time is reduced by avoiding the sampling operation.  $r_2$  is set to 80% and is adjustable to achieve different accuracy-latency tradeoffs.

## VI. EVALUATION

### A. Experiment Setup

We evaluate Parallel-SA on detection networks PointRCNN [30], 3DSSD [33], and IASSD [34] on the KITTI [29] and ONCE dataset [23], using mAP as the metric. Experiments are run on an NVIDIA RTX 2080Ti. The accuracy is calculated at the moderate difficulty level with 40 recall positions.

### B. Performance Evaluation

**KITTI Benchmark Results** Table I presents the detection accuracy comparison evaluated on KITTI [29], where Parallel-SA is evaluated with parallelism  $n=8$ . Networks that replace FPS with random sampling exhibit significant accuracy degradation. The average accuracy loss across PointRCNN [30], 3DSSD [33] and IASSD [34] is 13.6 in Car, 8.4 in Pedestrian and 13.1 in Cyclist. A naive parallel implementation splits  $N$  points into  $n$  partitions and samples  $K/n$  points from each

TABLE I: Accuracy Comparison.

	Method	Accuracy		
		Car	Pedestrian	Cyclist
PointRCNN [30]	Original	95.1	76.1	77.0
	Random sampling	81.2	66.9	63.8
	Naive parallel	92.9	73.2	72.8
	<b>Parallel-SA</b>	<b>95.1</b>	<b>76.4</b>	<b>77.1</b>
3DSSD [33]	Original	95.4	<b>78.1</b>	80.6
	Random sampling	78.4	68.7	66.8
	Naive parallel	92.8	73.5	74.3
	<b>Parallel-SA</b>	<b>95.3</b>	77.5	<b>81.0</b>
IASSD [34]	Original	95.5	78.8	78.2
	Random sampling	85.6	72.3	65.8
	Naive parallel	92.6	73.2	74.1
	<b>Parallel-SA</b>	<b>95.5</b>	<b>79.2</b>	<b>78.5</b>

partition. The average accuracy loss across PointRCNN [30], 3DSSD [33], and IASSD [34] is 2.6 in Car, 4.4 in Pedestrian, and 4.9 in Cyclist. Parallel-SA preserves global spatial uniformity in parallel set abstraction and achieves comparable accuracy to the original point-based detection networks.

TABLE II: Evaluation on ONCE

Method	Accuracy		Speedup
	Vehicle	Cyclist	
Original	48.2	25.3	-
Random Sampling	42.7	17.5	2.80×
Naive Parallel	43.1	24.6	2.70×
<b>Parallel-SA</b>	<b>46.2</b>	<b>27.4</b>	2.67×

**ONCE Benchmark Results** We evaluate our method on the ONCE dataset [23] using PointRCNN [30] as the baseline network. Other networks are not included due to the unavailable implementation specifications for the ONCE dataset. Table II shows that PointRCNN [30] with random sampling has 5.5 accuracy loss in Vehicle and 7.8 in Cyclist. PointRCNN [30] with naive parallel set abstraction exhibits 5.1 loss in Vehicle and 0.7 in Cyclist. In contrast, Parallel-SA achieves 2.67× speedup with minimal accuracy loss.

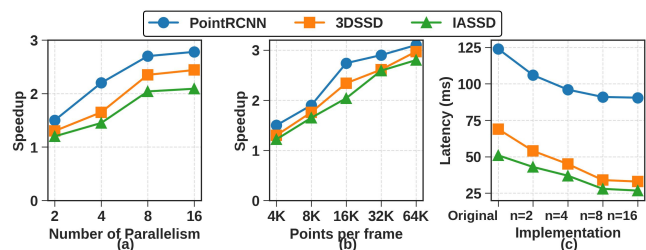


Fig. 8: Speedup of Parallel-SA (a) Speedup on set abstraction with different parallelism. (b) Speedup on set abstraction with varying input sizes. (c) Latency with different parallelism.

**Set Abstraction Speedup** Figure 8 (a) presents the speedup over the original set abstraction achieved by Parallel-SA with different parallelism. With parallelism increase from 2 to 8, Parallel-SA delivers speedups up to 2.74×, 2.35×, 2.04× for PointRCNN [30], 3DSSD [33], and IASSD [34]. However, increasing parallelism to 16 yields only marginal speedup, because the increased computational overhead of balanced partitioning negates parallelization’s performance gains. Therefore,

we selected  $n=8$  as the optimal value, as higher parallelism offers only marginal speedups at greater computational cost. Figure 8 (b) shows the speedup achieved by Parallel-SA with parallelism  $n=8$  when the input size grows from 4K points to 64K points per frame. The benefits of Parallel-SA become more significant with increased input size, achieving up to  $3.05\times$ ,  $2.97\times$ , and  $2.80\times$  speedup for PointRCNN [30], 3DSSD [33], and IASSD [34] when the number of points per frame is 64K. Set abstraction acceleration significantly reduces per-frame detection latency, as shown in Figure 8 (c). With  $n=8$  parallelism, Parallel-SA reduces latency from 124 to 90 ms for PointRCNN [30], 69 to 33 ms for 3DSSD [33], and 51 to 27 ms for IASSD [34]. Further increasing  $n$  to 16 offers minimal benefit due to partitioning overhead.

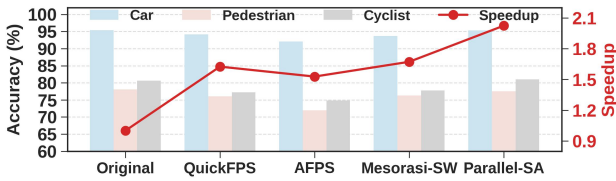


Fig. 9: Performance Comparison with QuickFPS [28], AFPS [35] and Mesorasi-SW [6].

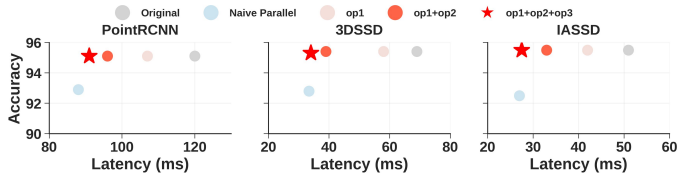


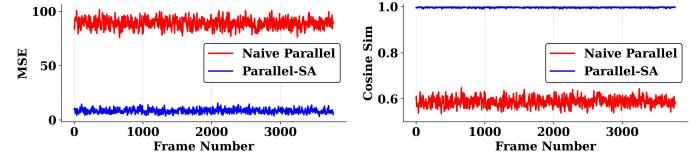
Fig. 10: The baseline **Naive Parallel** implements straight-forward parallel processing; **op1** enhances the networks with multi-scale sampling distribution estimation; **op1+op2** introduces distribution-aware balanced partitioning; **op1+op2+op3** introduces adaptive load balance refinement.

**Comparison with Related Work** We compare Parallel-SA with three existing approaches: QuickFPS [28], AFPS [35] and Mesorasi-SW [6]. Figure 9 presents the comparison results evaluated on 3DSSD [33]. Across the three classes, the average loss of QuickFPS [28] is 2.2, AFPS [35] is 5.1, and Mesorasi-SW [6] is 2.1, while the loss of Parallel-SA is only 0.1. Parallel-SA also achieves a higher speedup compared with the related works.

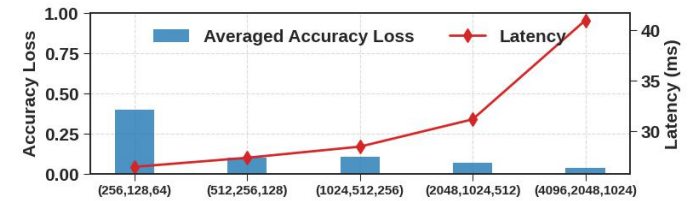
### C. Improvement Analysis

We evaluate the improvement of each proposed optimization technique for PointRCNN [30], 3DSSD [33], and IASSD [34]. The accuracy represents the Car class detection performance. Figure 10 demonstrates the results evaluated on KITTI [29] with parallelism  $n=8$ . By adding optimizations progressively, we evaluate the contributions of each stage: (1) The **Naive Parallel** implementation divides  $N$  points into  $n$  partitions and samples  $K/n$  points from each partition in parallel. (2) **op1** enhances the original networks with multi-scale sampling

distribution estimation. It employs fixed boundaries in the partitioning steps. (3) **op1+op2** extends the framework with distribution-aware balanced partitioning. This partitioning strategy enhances system efficiency while maintaining model accuracy. (4) **op1+op2+op3** further optimizes the system through adaptive load balance refinement.



(a) Distribution approximation error for each frame.



(b) Accuracy loss and latency with different downsizing scales.

Fig. 11: Analysis of distribution approximation.

### D. Analysis of Distribution Approximation

**Distribution Approximation Error** We evaluate the distribution approximation with  $n=8$  and  $(m_1, m_2, m_3)=(1024, 512, 256)$ . The ground truth distribution is obtained by applying FPS to sample  $K$  points from the point cloud and assigning them to partitions  $k_{gt_1}, \dots, k_{gt_n}$ . The naive parallel implementation samples  $k_{n_i}=K/n$  points for each partition. Parallel-SA estimates the optimal sampling number of each partition  $k_{m_1}, \dots, k_{m_n}$ . These distributions are converted to vectors for quantitative comparison using mean squared error (MSE) and cosine similarity. As shown in Figure 11a, Parallel-SA achieves an average MSE of 8.14 and cosine similarity of 0.9963, while the naive parallel implementation results in an average MSE of 88.99 and cosine similarity of 0.5868.

**Latency Analysis** We evaluate the performance of Parallel-SA with distribution approximation at different downsizing scales  $(m_1, m_2, m_3)$ . At each scale, we randomly downsize the input  $N$  points to the specified scale and then applying FPS with a sampling rate of 25%. As shown in Figure 11b, small scales (256,128,64) lead to larger accuracy loss due to insufficient estimation samples for reliable approximation. Increasing the number of samples improves distribution approximation quality, leading to reduced accuracy loss. However, scales larger than (2048, 1024, 512) result in significantly increased latency while providing only marginal accuracy improvements.

## VII. CONCLUSION

As a core feature extractor for many state-of-the-art detection networks, set abstraction has a high computational cost. We propose Parallel-SA to convert it from a sequential process to a parallel process. With well-designed partitioning and estimation of apply, Parallel-SA can achieve an average speedup of  $2.38\times$  in set abstraction with minimal accuracy loss.

## REFERENCES

- [1] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, Deep learning for 3D point clouds: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 12, pp. 4338–4364, 2020.
- [2] J. Mao, S. Shi, X. Wang, and H. Li, 3D object detection for autonomous driving: A comprehensive survey," *International Journal of Computer Vision*, vol. 131, no. 8, pp. 1909–1963, 2023.
- [3] R. Qian, X. Lai, and X. Li, 3D object detection for autonomous driving: A survey," *Pattern Recognition*, vol. 130, p. 108796, 2022.
- [4] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, PointNet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in Neural Information Processing Systems*, vol. 30, 2017.
- [5] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 652–660.
- [6] Y. Feng, B. Tian, T. Xu, P. Whatmough, and Y. Zhu, Mesorasi: Architecture support for point cloud analytics via delayed-aggregation," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 1037–1050.
- [7] W. Liu, J. Sun, W. Li, T. Hu, and P. Wang, Deep learning on point clouds and its application: A survey," *Sensors*, vol. 19, no. 19, p. 4188, 2019.
- [8] 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.
- [9] Q. Wang, J. Chen, J. Deng, and X. Zhang, 3D-CenterNet: 3D object detection network for point clouds with center estimation priority," *Pattern Recognition*, vol. 115, p. 107884, 2021.
- [10] M. Lee, S. Park, H. Kim, M. Yoon, J. Lee, J. W. Choi, N. S. Kim, M. Kang, and J. Choi, SPADE: Sparse Pillar-based 3D Object Detection Accelerator for Autonomous Driving," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 454–467.
- [11] H. Lu and H. Shi, Deep learning for 3D point cloud understanding: A survey," *arXiv preprint arXiv:2009.08920*, 2020.
- [12] P. Zhao, G. Yuan, Y. Cai, W. Niu, Q. Liu, W. Wen, B. Ren, Y. Wang, and X. Lin, Neural pruning search for real-time object detection of autonomous vehicles," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2021.
- [13] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, RandLA-Net: Efficient semantic segmentation of large-scale point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11108–11117.
- [14] H. Tang, S. Yang, Z. Liu, K. Hong, Z. Yu, X. Li, G. Dai, Y. Wang, and S. Han, TorchSparse++: Efficient Training and Inference Framework for Sparse Convolution on GPUs," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '23, ACM*, 2023, pp. 225–239.
- [15] J.-F. Zhang and Z. Zhang, Point-X: A spatial-locality-aware architecture for energy-efficient graph-based point-cloud deep learning," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1078–1090.
- [16] Tang, D., Sun, X., Guan, N., Kuo, T. W., & Xue, C. J. (2022, December). pLPAQ: Accelerating LPAQ Compression on FPGA. In *2022 International Conference on Field-Programmable Technology (ICFPT)* (pp. 1-6).
- [17] Tang, D., Mao, Y., Wang, W., Guan, N., Kuo, T. W., & Xue, C. J. (2025). *DAWN: Accelerating Point Cloud Object Detection via Object-Aware Partitioning and 3D Similarity-Based Filtering*. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)* (pp. 1-7).
- [18] Tang, D., Wang, W., Mao, Y., Yu, J., Kuo, T. W., & Xue, C. J. (2024). *STEM: Streaming-Based FPGA Acceleration for Large-Scale Compactions in LSM KV*. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)* (pp. 3893-3905).
- [19] Wang, W., Mao, Y., Tang, D., Du, H., Guan, N., & Xue, C. J. *When Compression Meets Model Compression: Memory-Efficient Double Compression for Large Language Models*. In *Findings of the Association for Computational Linguistics: EMNLP 2024* (pp. 16973–16983). Association for Computational Linguistics, Miami, Florida, USA.
- [20] H. Yoon and J.-J. Kim, Efficient sampling and grouping acceleration for point cloud deep learning via single coordinate comparison," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, IEEE, 2023, pp. 1–9.
- [21] C.-C. Peng and A.-C. Chang, "Unorganized 3D point clouds denoising and sharpening," in *2024 International Conference on Control, Automation and Diagnosis (ICCAD)*, 2024, pp. 1–6. doi: 10.1109/ICCAD60883.2024.10553957.
- [22] Z. Song, H. Lu, G. Li, L. Jiang, N. Jing, and X. Liang, "PRADA: Point cloud recognition acceleration via dynamic approximation," in *Proceedings of the 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6.
- [23] J. Mao, M. Niu, C. Jiang, X. Liang, Y. Li, C. Ye, W. Zhang, Z. Li, J. Yu, C. Xu, et al., One Million Scenes for Autonomous Driving: ONCE Dataset," *arXiv preprint arXiv:2106.11037*, 2021.
- [24] Z. Song, L. Liu, F. Jia, Y. Luo, C. Jia, G. Zhang, L. Yang, and L. Wang, Robustness-aware 3D object detection in autonomous driving: A review and outlook," *IEEE Transactions on Intelligent Transportation Systems*, 2024.
- [25] A. A. M. Muzahid, H. Han, Y. Zhang, D. Li, Y. Zhang, J. Jamshid, and F. Sohel, Deep learning for 3D object recognition: A survey," *Neurocomputing*, p. 128436, 2024.
- [26] T. Xu, B. Tian, and Y. Zhu, Tigris: Architecture and algorithms for 3D perception in point clouds," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 629–642.
- [27] R. Pinkham, S. Zeng, and Z. Zhang, QuickNN: Memory and performance optimization of kd tree based nearest neighbor search for 3D point clouds," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 180–192.
- [28] M. Han, L. Wang, L. Xiao, H. Zhang, C. Zhang, X. Xu, and J. Zhu, QuickFPS: Architecture and algorithm co-design for farthest point sampling in large-scale point clouds," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 4011–4024, 2023.
- [29] A. Geiger, P. Lenz, and R. Urtasun, Are we ready for autonomous driving? The KITTI vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361.
- [30] S. Shi, X. Wang, and H. Li, PointRCNN: 3D object proposal generation and detection from point cloud," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 770–779.
- [31] H. Wu, J. Deng, C. Wen, X. Li, C. Wang, and J. Li, CasA: A cascade attention network for 3-D object detection from LiDAR point clouds," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–11, 2022.
- [32] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, PV-RCNN: Point-voxel feature set abstraction for 3D object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10529–10538.
- [33] Z. Yang, Y. Sun, S. Liu, and J. Jia, 3DSSD: Point-based 3D single stage object detector," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11040–11048.
- [34] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 18953–18962.
- [35] J. Li, J. Zhou, Y. Xiong, X. Chen, and C. Chakrabarti, An adjustable farthest point sampling method for approximately-sorted point cloud data," in *2022 IEEE Workshop on Signal Processing Systems (SiPS)*, 2022, pp. 1–6.
- [36] M. Han, L. Wang, L. Xiao, H. Zhang, C. Zhang, X. Xie, S. Zheng, and J. Dong, FuseFPS: Accelerating Farthest Point Sampling with Fusing KD-tree Construction for Point Clouds," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024, pp. 238–243.
- [37] X. Liu, Z. Song, G. Dai, G. Li, C. Xiao, Y. Xiang, D. Kong, K. Xu, and X. Liang, FusionArch: A Fusion-Based Accelerator for Point-Based Point Cloud Neural Networks," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–6.