Real-Time Fast Channel Clustering for LiDAR Point Cloud

Xiao Zhang, *Graduate Student Member, IEEE*, and Xinming Huang, *Senior Member, IEEE*

***Abstract*—LiDAR sensors can produce point clouds with precise 3D depth information that is essential for autonomous vehicles and robotic systems. As a perception task, point cloud clustering algorithms can be applied to segment the points into object instances. In this brief, we propose a novel, hardware-friendly fast channel clustering (FCC) algorithm that achieves state-of-the-art performance when evaluated using KITTI panoptic segmentation benchmark. Furthermore, an efficient, pipeline hardware architecture is proposed to implement the FCC algorithm on an FPGA. Experiments show that the hardware design can process each LiDAR frame with 64 channels, 2048 horizontal resolution at various point sparsity in 1.93 ms, which is more than 471.5 times faster than running on the CPU. The code will be released to the public via GitHub.**

***Index Terms*—Point cloud, LiDAR, clustering, real-time processing, FPGA.**

I. Introduction

POINT cloud generated by light detection and ranging (LiDAR) sensor and its related processing technology have attracted extensive research interests in recent years. LiDAR sensor is deployed in either outdoor or indoor scenarios to obtain precise depth information. A typical LiDAR system can sense the objects with an estimated range of 80 to 100 meters at 10 Hz, and it delivers abundant 3D information of the surroundings. For example, in KITTI dataset [1] a LiDAR frame includes various objects such as pedestrians, cyclists, cars, buildings, trees, ground, etc. Segmenting and classifying these objects efficiently and accurately is a crucial task.

First of all, point cloud clustering can be applied for instance segmentation [2], [3], [4]. Moreover, clustering algorithms can be included as part of a high-level perception system. For example, an adaptive euclidean clustering [5] is used to generate the candidates for an SVM-based human classifier. Clustering algorithms have also been incorporated to the deep learning based perception models. For point cloud semantic segmentation [6], it employed a clustering [4] step to capture the objects before feeding to the neural network for classification. In a fusion network [7], clustering algorithm was implemented to detect the object in the point cloud first. Projecting the cluster bonding box onto the image, a 2D region proposal generation layer is built to fuse the information from both LiDAR point cloud and camera image data into the convolutional neural network (CNN). Besides pre-processing, clustering can also used for post-processing, such as object instancing after semantic segmentation [8].

The size of a LiDAR point cloud frame is determined by its vertical and horizontal angular resolutions. A typical 64 lines LiDAR produces a frame every 100 ms, each containing about 100k points, which urges for a large storage and high computation capacity for processing. As the LiDAR resolution grows, it poses a significant challenge for an on-board embedded platform to meet the real-time constrain. The existing LiDAR clustering algorithms take about tens of milliseconds [2], [4] to process each frame on a CPU or GPU. Since clustering is only a small part of the overall perception system, it needs to be accelerated to leave more time for the main detection or classification tasks. Moreover, the popular LiDAR clustering algorithms are not hardware friendly since they often perform unbalance data query as well as random data access when searching for the nearest neighboring points.

In this brief, we propose a range image based, hardware-friendly clustering algorithms and its pipeline architecture. Our contributions are listed as follows.

(1) We propose a novel, range image based, three-pass grouping algorithm called fast channel clustering (FCC). To address the problem of over-segmentation caused by missing points, we design a scalable connection filter to search the local neighborhood for inter-channel connections. A merge table is used to store the connected labels efficiently.

(2) We first implement and verify the range image based LiDAR point cloud clustering method by targeting on a Zynq-7000 FPGA. Combining a line buffer with a unique sort network that enables fully parallel calculations of Euclidean distances and unique sorting among the neighboring points, the FPGA implementation achieves 2 to 3 orders of magnitude of speed-up over CPU at run-time.

II. Related Work

Clustering Methods: Clustering is a classic topic in data mining for grouping the data elements by similarity. However, traditional clustering methods primarily focus on the points in a 2D plane or other multi-dimensional data. In contrast, point cloud data can be considered as 3D (X, Y, Z) points,

Manuscript received 15 April 2022; revised 1 June 2022; accepted 16 June 2022. Date of publication 22 June 2022; date of current version 26 September 2022. This work was supported in part by U.S. NSF under Grant 2006738, and in part by MathWorks. This brief was recommended by Associate Editor V. Paliouras. *(Corresponding author: Xiao Zhang.)*

The authors are with the Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, Worcester, MA 01609 USA (e-mail: xzhang25@wpi.edu; xhuang@wpi.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSII.2022.3185228.

Digital Object Identifier 10.1109/TCSII.2022.3185228

sometimes with extra attributes, i.e., norm, RGB, etc. In this case, not all existing clustering methods can efficiently adapt to the task of LiDAR point clouds. In general, the existing point cloud clustering methods can be classified into two categories, including point query in 3D space and point query in a range image.

For 3D space query clustering, typically, the 3D points are fitted into storage structures like kd-tree or oct-tree to optimize the nearest neighbor search procedure. For instance, using the Euclidean distance is a popular approach. In [9], the authors developed a radially bounded nearest neighbor (RBNN) algorithm. A novel ground segmentation algorithm was proposed in [10], where non-ground points were clustered with voxelized Euclidean neighbors. In [11], researchers provided a probabilistic framework to incorporate not only the Euclidean spatial information but also the temporal information from consecutive frames. In [5], an adaptive threshold selecting method is presented to deal with the changes of LiDAR sparsity along with the distance, resulting an improved performance for the application of human detection.

Range image based clustering is applicable to LiDAR point clouds since points from a LiDAR frame can be mapped as range image pixels by spherical projection according to the sensor mechanism. For those organized point clouds emitted from a sole LiDAR sensor, utilizing the natural range image representation allows the algorithm to work in linear complexity of the point cloud size. In [12], the angle formed by two adjacent laser beams is considered to justify if they belong to the same object. Other conditions, like multiple distance thresholds, are investigated in [4]. Those range image methods usually borrow ideas from connected-component labeling (CCL) algorithms [4], [13]. CCL is a graph algorithm used in computer vision to detect connected regions in a binary digital image [14]. Due to the inherent difference between binary images and LiDAR range images, authors of [4], [13] modified existing CCL methods, such as two-pass CCL [15] or run-based CCL [16], to deal with the LiDAR data. For more details, the survey on LiDAR point cloud clustering [17] is worth reaching.

Clustering on FPGA: The existing hardware architecture for clustering were mainly targeted for data mining. In [18], the authors developed an FPGA architecture for processing micro-array data. A tree-based structure was implemented on FPGA to accelerate the processing in [19]. In addition, k-means cluster was popular in image processing and several works were aimed for hardware acceleration. For color images, a design by [20] met the real-time requirement. A multi-core structure on FPGA was proposed in [21]. However, k-means clustering method is not competitive when applied to LiDAR point clouds compared to the methods we reviewed above.

III. Fast Channel Clustering

Fast channel clustering is a range image based method following the principle of CCL. The main objective of FCC is to provide a method with hardware efficiency and low latency while maintaining high accuracy for LiDAR point cloud tasks. An example of segmentation result is shown in Fig. 1.

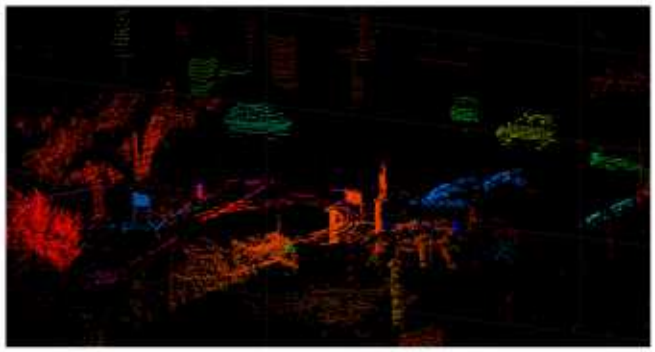

Fig. 1. The segmentation result of Fast Channel Clustering (FCC).

We assume that a ground removal procedure is applied before the point cloud is fed to the clustering algorithm as a range image. Then, a three-pass FCC algorithm is applied to group points based on object instances. The first pass is the intra-channel grouping based on a horizontal distance threshold th_H. The second pass is inter-channel grouping using a connection filter, which merges the channel-wise clusters within the vertical distance threshold th_V. The third pass is to update the labelling table for output. The following subsections describe our three-pass algorithm in detail.

A. Pass One: Intra-Channel Grouping

As shown in Fig. 2, the first pass aims to group the points from the same object in every LiDAR scanning channel. We check the Euclidean distance between every two adjacent valid points, and create a new channel-wise cluster if the distance is larger than th_H or extend the current one if not.

According to the LiDAR scanning mechanism, each channel is a ring in 3D space. Hence, a *ringCheck* strategy is designed to guarantee the continuity by checking the distance between the last point and the first point of a channel. If the *ringCheck*(i) is true, these two channel-wise cluster will be merged. Furthermore, the *updateGlobalLabel* function adds the *maxLabel* of the above channel to give every cluster a unique label in the global view. More details about intra-channel grouping can refer to Algorithm 1.

B. Pass Two: Inter-Channel Grouping

Pass two takes charge of inter-channel grouping. In this part, a filter-like connection check is designed to merge the channel-wise clusters. The connection filter determines the connectivity among the current cluster and the clusters in adjacent channels. Due to the missing point issues, points belonging to the same object may be separated in several nonadjacent channels, causing the over-segment problem. In this case, the filter size can determine the searching space in the local range image area to address the issue.

To save the memory and avoid iterative passes of updating labels, we do not directly update point labels in the range image during the filtering. Instead, we use a merge table to record the merging results. In Fig. 2, the address of the merge table indicates the initial channel-wise label (from pass-one), and the value inside marks the merged result.

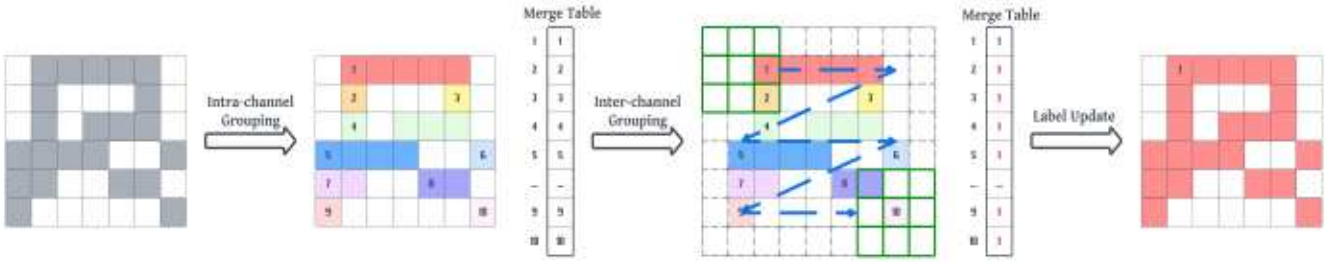

Fig. 2. The steps of fast channel clustering with a 3-by-3 connection filter.

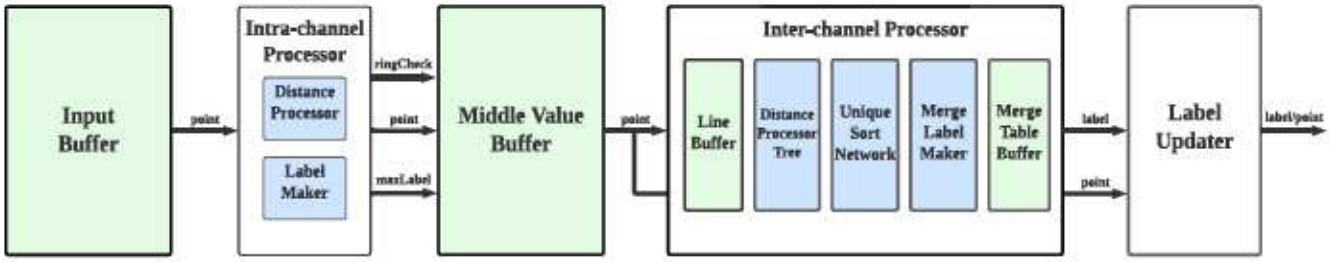

Fig. 3. **Hardware Architecture** The modules in green represent different buffers. The modules in blue indicate the key sub-functional modules. The modules in white denote the principle functional modules controlled by FSM.

Algorithm 1 Pass One: Intra-Channel Grouping

Input : Range image $I_{xyz}(h, w, 3)$
Valid mask of range image $m(h, w, 1)$
Input : Intra-channel threshold th_H
Output: Range image with x, y, z, valid, global label I_{xyzvl}

$maxLabel \leftarrow zeros(h, w)$
$ringCheck \leftarrow zeros(h, w)$
$I_{xyzvl}(h, w, 5) = concatenate([I_{xyz}, m\ , zeros(h, w)],$ "$axis = 3$")
for $i = 1 : h$ **do**
  $firstPoint = I_{xyzvl}(i, 1, :)$
  $neighborPoint = I_{xyzvl}(i, 1, :)$
  for $j = 1 : w$ **do**
    $currentPoint = I_{xyzvl}(i, j, :)$
    $valid = currentPoint(4)$
    if ***not*** *valid* **then**
      continue
    end
    $d =$**dist**$(currentPoint, neighborPoint)$
    if $d > th_H$ **then**
      $maxLabel(i) + +$
    end
    $I_{xyzvl}(i, j, 5) = maxLabel(i)$
    $neighborPoint = currentPoint$
    $lastPoint = currentPoint$
  end
  $d =$**dist**$(firstPoint, lastPoint)$
  if $d \leq th_H$ **then**
    $ringCheck(i) = true$
  end
end
updateGlobalLabel$(I_{xyzvl}, maxLabel, ringCheck)$

The filter center steps through every valid point on the range image. During the checking procedure, the filter checks the distances between the center point and each valid point above the current channel. If the distance is smaller than th_V, these two clusters are connected, and the clusters with a large label number are merged into the one with the smallest label. The *labelsToMerge* cached the connected labels, when the filter passes the points in a single channel-wise cluster. Finally, a merging process is triggered and the cache is cleared before the connection filter slides to the next cluster. Nore that the connectivity is transferable. Consequently, the label merging will ongoing until the filter passes through all points in the same object and merges them into the uniform label. More details are available in Algorithm 2.

C. Pass Three: Label Update

As shown in Fig. 2, the final pass is just to consolidate the merge table through lookup and each channel-wise cluster is updated with the final label.

IV. Hardware Architecture

As shown in Fig. 3. The point cloud is collected in the input buffer. Our hardware design has three main components corresponding to the three-pass algorithm.

A. Intra-Channel Processor

Intra-channel processors perform the channel-wise grouping. We implement a two-step point-wise pipeline. In the first step, the distance processor computes the distance between the adjacent valid points in the same channel. In the second step, the label maker assigns a channel-wise label for each point. The *ringCheck* and *maxLabel* are stored along with the point cloud frame with the channel-wise label in the middle-value buffer. All channels are processed in parallel and channel-wise labels are unique.

B. Inter-Channel Processor

A line buffer is placed to provide the pixel values as the connection filter slides through the entire range image

Algorithm 2 Pass Two: Inter-Channel Grouping

Input : Range image with global label $I_{xyzvl}(h, w, 5)$
Input : Inter-channel threshold th_V
Connection filter size f_{size}
Output: Merge table map the after and before merging label T_{merge}

T_{merge} = **InitTable**($max(I_{xyzvl}(:, :, 5))$)
$f_{width} = (f_{size} - 1)/2$
$padding(I_{xyzvl}, f_{width})$
$label\_past = 1$
$labelsToMerge$ = []
for $i = 1 : h$ **do**
  for $j = 1 : w$ **do**
    $f = zeros(f_{size}, f_{size}, 5)$
    for $m = 1 : f_{size}$ **do**
      for $n = 1 : f_{size}$ **do**
        offset$Y = i + m - 1$; offset$X = j + n - 1$
        $f(m, n, :) = I_{xyzvl}$(offset$Y$, offset$X$, :)
      end
    end
    $corePoint = f(f_{width} + 1, f_{width} + 1, :)$
    $valid = corePoint(4)$, $label = corePoint(5)$
    if ***not*** *valid* **then**
      continue
    end
    findMergeLabels($labelsToMerge$, f, th_V)
    if $label \neq label\_past$ ***or*** $j = w$ **then**
      updateMergeTable(T_{merge}, $labelsToMerge$)
      $labelsToMerge$ = []
      $label\_past = label$
    end
  end
end

Procedure findMergeLabels ($labelsToMerge$, f, th_V)
  $lines = f(1 : f_{width}, :, :)$
  $corePoint = f(f_{width} + 1, f_{width} + 1, :)$
  for *line* ***in*** *lines* **do**
    for *point* ***in*** *line* **do**
      d =**dist** (*point*, *corePoint*)
      if $d < th_H$ **then**
        $labelsToMerge$.pushback($T_{merge}(corePoint(5))$
        $labelsToMerge$.pushback($T_{merge}(point(5))$
      end
    end
  end
end
Procedure updateMergeTable (T_{merge}, $labelsToMerge$)
  $labelsToMerge = unique(labelsToMerge)$
  $L_{target} = labelsToMerge(1)$
  for L ***in*** $labelsToMerge$ **do**
    $T_{merge}(L) = T_{merge}(L_{target})$
  end
end

in a raster scan pattern. The distance processor tree calculates the distances from the center point to all valid points within the filter but above the current channel. Then, the unique sort networks, which also include a duplicate detector, collect the unique to-merge labels and passes them to the merge label maker. After the filter window slides through the points in the same channel-wise cluster, the *labelsToMerge* cached in the FIFO can update the T_{merge} in the merge table buffer.

TABLE I
FPGA RESOURCE USAGE FOR DIFFERENT FILTER SIZES

Architecture	Image-based Cluster			
Component	LUTs	Registers	DSP	BRAM
3x3 FCC	4810(2.20%)	5477(1.25%)	15(1.67%)	359.5(66.15%)
5x5 FCC	6591(3.02%)	7482(1.71%)	36(4.00%)	361.5(66.33%)
7x7 FCC	9304(4.26%)	10222(2.34%)	69(7.67%)	362.5(66.51%)

A larger filter size requires more DSP slices in the distance processor tree in order to calculate the distances to more points in parallel. A larger filter also calls for larger sort networks, which result more LUT and register usage and slower clock speed. We recommend a filter size up to 7x7 as the better trade-off between performance, clock constraint and resource usage.

TABLE II
CLUSTERING PERFORMANCE COMPARISON USING SEMANTICKITTI PANOPTIC SEGMENTATION

Methods	Settings	*PQ*
Depth Cluster	$\theta = 10°$	55.2
Scan-line Run [4]	th_{run}, th_{merge}= 0.5m, 1.0m	54.1
*Scan-line Run [17]	th_{run}, th_{merge}= 0.5m, 1.0m	57.2
FCC	th_H, th_V= 0.5m, 1.0m, $f_{size} = 3 \times 3$	51.7
FCC	th_H, th_V= 0.5m, 1.0m, $f_{size} = 5 \times 5$	56.1
FCC	th_H, th_V= 0.5m, 1.0m, $f_{size} = 7 \times 7$	56.8
FCC	th_H, th_V= 0.5m, 1.0m, $f_{size} = 9 \times 9$	57.1
FCC	th_H, th_V= 0.5m, 1.0m, $f_{size} = 11 \times 11$	**57.3**

The result of the Scan-line Run is from the original method [4], and the *Scan-line Run outcome is based on the improvement method with make-up search optimization [17]. *PQ* [22] is a metric to evaluate the panoptic segmentation result in terms of the object instance ability.

C. Label Updater

Label updater maps the channel-wise labels to the post-merging labels by reading the values at the corresponding address in the merge table buffer.

D. Quantization and Data Storage

A fixed-point quantization strategy is applied for efficient use of the on-chip block RAM. Distance value is quantized to a 16-bit signed value with an 8-bit fraction and 7-bit integer since the LiDAR sensing range is within 100 meters. The cluster labels are stored as 15-bit unsigned integers.

E. Simulation and Implementation

The hardware design is implemented using MATLAB Simulink HDL Coder and Vision HDL Toolbox. The input size is 64-by-2048 LiDAR frames. At last, the entire HDL model is targeted on the Xilinx ZC706 board with the XC7Z045 FPGA device for synthesis and implementation in Vivado. The synthesis result is provided as in Table I. All three designs have passed the 203.6 MHz clock constraint.

V. EXPERIMENT

A. Performance Evaluation

In our previous work [17], we proposed to evaluate the performance of LiDAR point cloud clustering accurately and fairly in a quantitative matter using a panoptic segmentation task from SemanticKITTI that includes two parts - semantic segmentation and object instance. Using the same semantic checkpoint, the object instance accuracy of different clustering methods can be compared according to the panoptic results.

TABLE III
ALGORITHM INFERENCE TIME ON CPU AND FPGA

Methods	Inference Time
*FCC 3x3	0.91 s
*FCC 5x5	1.63 s
*FCC 7x7	2.77 s
*FCC 9x9	4.17 s
*FCC 11x11	6.29 s
FCC 7x7 on FPGA	**1.93 ms**

Average Inference time of one LiDAR frame in sequence 8. *Implementations run on a single thread of i5-4590 CPU @ 4.00GHz with the environment set of windows subsystem for Linux (WSL) vs. the ZC706 FPGA @ 203.6MHz (implemented with filter size of 7x7)

Hereby, we follow the same method and benchmark to evaluate the performance of the proposed FCC algorithm with the result listed in Table II.

As explained in Section III, the filter size is related to the local search area during the inter-channel grouping. Therefore, a larger filter size provides a better connection detection of channel-wise components in the same object. In some cases, larger filter size is necessary. The missing points in the LiDAR point cloud can cause over-segmentation, especially when the object is close to the LiDAR sensor and the object is stretched on the range image. From Table II, the filter size increase results in better performance. Eventually, PQ converges to around 57. The method provides state-of-the-art performance with an 11-by-11 filter size, surpassing depth clustering and SLR.

B. Processing Time on Hardware

We provide the run time of our FCC algorithm on CPU and the latency on FPGA in Table III above. Our FPGA design is about 471.5 (3x3 case) to 1435.2 (7x7 case) times faster than the algorithm running on the CPU, varying on different filter sizes. The CPU run time is relatively slow since we realized the algorithm in Python without multi-thread optimization. If compared with the run-time of other existing systems [2], [4], our FPGA implementation is still 8.2x faster on average.

VI. CONCLUSION

We present a novel, hardware-friendly, range image-based LiDAR point cloud clustering algorithm, namely fast channel clustering. By intra-channel grouping and inter-channel grouping, the method segments and merges the cloud in horizontal and vertical spaces. A scalable connection filter structure is proposed for inter-channel grouping to solve the missing point issues. Additionally, the merge table avoids the multi-pass update for merging the connected labels. As a result, the object instance performance can outperform the state-of-the-art methods. Finally, an efficient hardware design is implemented and verified on an FPGA for real-time point cloud processing. The FPGA design provides a 471.5 to 1435.2 times speedup to the CPU run-time of the FCC.

REFERENCES

[1] J. Behley *et al.*, "SemanticKITTI: A dataset for semantic scene understanding of LiDAR sequences," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 9297–9307.

[2] I. Bogoslavskyi and C. Stachniss, "Efficient online segmentation for sparse 3D laser scans," *J. Photogrammetry Remote Sens. Geoinf. Sci.*, vol. 85, no. 1, pp. 41–52, 2017.

[3] H. Yang, Z. Wang, L. Lin, H. Liang, W. Huang, and F. Xu, "Two-layer-graph clustering for real-time 3D LiDAR point cloud segmentation," *Appl. Sci.*, vol. 10, no. 23, p. 8534, 2020.

[4] D. Zermas, I. Izzat, and N. Papanikolopoulos, "Fast segmentation of 3D point clouds: A paradigm on LiDAR data for autonomous vehicle applications," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2017, pp. 5067–5073.

[5] Z. Yan, T. Duckett, and N. Bellotto, "Online learning for 3D LiDAR-based human detection: Experimental analysis of point cloud clustering and classification methods," *Auton. Robots*, vol. 44, no. 2, pp. 147–164, 2020.

[6] X. Kong, G. Zhai, B. Zhong, and Y. Liu, "PASS3D: Precise and accelerated semantic segmentation for 3D point cloud," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2019, pp. 3467–3473.

[7] X. Zhao, P. Sun, Z. Xu, H. Min, and H. Yu, "Fusion of 3D LiDAR and camera data for object detection in autonomous vehicle applications," *IEEE Sensors J.*, vol. 20, no. 9, pp. 4901–4913, May 2020.

[8] Y. Zhao, X. Zhang, and X. Huang, "A divide-and-merge point cloud clustering algorithm for LiDAR panoptic segmentation," 2021, *arXiv:2109.08224*.

[9] K. Klasing, D. Wollherr, and M. Buss, "A clustering method for efficient segmentation of 3D laser data," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2008, pp. 4043–4048.

[10] B. Douillard *et al.*, "On the segmentation of 3D LiDAR point clouds," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 2798–2805.

[11] D. Held, D. Guillory, B. Rebsamen, S. Thrun, and S. Savarese, "A probabilistic framework for real-time 3D segmentation using spatial, temporal, and semantic cues," in *Proc. Robot. Sci. Syst.*, vol. 12, Jun. 2016, p. 24, doi: 10.15607/RSS.2016.XII.024. [Online]. Available: http://www.roboticsproceedings.org/rss12/p24.html

[12] I. Bogoslavskyi and C. Stachniss, "Fast range image-based segmentation of sparse 3D laser scans for online operation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2016, pp. 163–169.

[13] F. Hasecke, L. Hahn, and A. Kummert, "FLIC: Fast LiDAR image clustering," in *Proc. ICPRAM*, 2021, pp. 25–35.

[14] L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao, "The connected-component labeling problem: A review of state-of-the-art algorithms," *Pattern Recognit.*, vol. 70, pp. 25–43, Oct. 2017.

[15] K. Wu, E. Otoo, and K. Suzuki, "Optimizing two-pass connected-component labeling algorithms," *Pattern Anal. Appl.*, vol. 12, no. 2, pp. 117–135, 2009.

[16] L. He, Y. Chao, and K. Suzuki, "A run-based two-scan labeling algorithm," *IEEE Trans. Image Process.*, vol. 17, no. 5, pp. 749–756, May 2008.

[17] Y. Zhao, X. Zhang, and X. Huang, "A technical survey and evaluation of traditional point cloud clustering methods for LiDAR panoptic segmentation," 2021, *arXiv:2108.09522*.

[18] H. M. Hussain, K. Benkrid, H. Seker, and A. T. Erdogan, "Fpga implementation of k−means algorithm for bioinformatics application: An accelerated approach to clustering microarray data," in *Proc. IEEE NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, 2011, pp. 248–255.

[19] F. Winterstein, S. Bayliss, and G. A. Constantinides, "FPGA-based k-means clustering using tree-based data structures," in *Proc. IEEE 23rd Int. Conf. Field Program. Logic Appl.*, 2013, pp. 1–6.

[20] T. Saegusa and T. Maruyama, "An FPGA implementation of real-time k-means clustering for color images," *J. Real-Time Image Process.*, vol. 2, no. 4, pp. 309–318, 2007.

[21] J. Canilho, M. Vestias, and H. Neto, "Multi-core for k-means clustering on FPGA," in *Proc. IEEE 26th Int. Conf. Field Program. Logic Appl. (FPL)*, 2016, pp. 1–4.

[22] J. Behley, A. Milioto, and C. Stachniss, "A benchmark for LiDAR-based panoptic segmentation based on KITTI," 2020, *arXiv:2003.02371*.