MoC: A Morton-Code-Based Fine-Grained Quantization for Accelerating Point Cloud Neural Networks

Xueyuan Liu, Zhuoran Song*, Hao Chen, Xing Li, Xiaoyao Liang
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China

ABSTRACT

Point Cloud Neural Network (PCNN) plays an essential role in various 3D applications, with some of them even being time-sensitive and safety-critical. However, the large scale of unordered points with lengthy features results in heavy computational workloads, making them far from real-time processing. To address this challenge, we propose MoC, a Morton-code-based fine-grained quantization for accelerating PCNNs. Specifically, we utilize Morton code to capture the spatial locality among points. Then, we gather nearby points with similar features into a region. Considering the similarity in features of nearby points, we propose to decompose features into base and offsets, where the offsets fall within a narrow range. Building upon this, we introduce a two-level mixed-precision quantization. In the first level, we quantize offsets with low precision, while keeping the base in high precision to ensure accuracy. For the second level, noticing the different data distribution of offsets across various regions, we employ two types of low precision at the region level, which provides opportunities to further accelerate feature computations. To support our algorithm, we design a hardware architecture that parallelizes the Morton code path with the critical path. In our extensive experiments on various datasets, our algorithm-architecture co-designed method demonstrates 12×, 6.3×, 4.7×, 3.8×, 3.4× and 2.8× speedup and 19.3×, 9.7×, 6.0×, 5.2×, 4.6× and 4.1× energy savings over CPU, Server and Edge GPUs, state-of-the-art ASICs (*incl.* PointAcc, MARS, PRADA) with negligible accuracy loss.

1 INTRODUCTION

The point cloud is a collection of unordered points scattered in 3D space, representing a sampling of the surfaces of physical objects and 3D scenes. PCNNs are introduced to extract point features for various backend networks, which play an essential role in autonomous driving, Augmented Reality (AR), etc. Notably, the lengthy feature dimensions lead to extremely heavy computational workloads, posing a performance bottleneck and making PCNNs far from real-time processing.

Numerous previous works have recognized the challenges associated with feature computation. For example, Mesorasi [4] proposes a "delayed aggregation" algorithm that alters the execution order between KNN and feature computation. It reduces the Multiply-Accumulate (MAC) operations in the MLP by calculating each point feature only once and eliminating the redundancy introduced by aggregation. This approach diminishes the number of replicated points at the cost of accuracy loss. PRADA [13] presents a "dynamic approximation" algorithm that classifies the points into key and non-key ones. The algorithm performs feature computation only once for the key points. In particular, it unifies nearby centroids as key points and skips sharing neighbors located in the intersection of nearby centroids by directly copying their features to restore. This method essentially reduces the points to be calculated, shown in Fig. 1. To summarize, both algorithms alleviate the computational workloads from the aspect of point numbers, but they overlook the contribution of features themselves to the computing results.

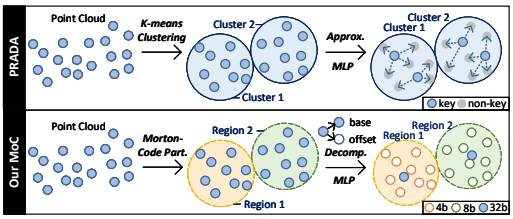

Figure 1: An overview of our MoC Algorithm.

We commence our investigation from the data itself based on the fact that points located in close spatial proximity exhibit similar features. To identify such similarity, a straightforward method is clustering points using K-means, which has been employed by PRADA. However, the K-means clustering is an expensive iterative algorithm that undergoes numerous iterations before reaching convergence. To capture the spatial locality of points while minimizing the grouping overhead, we employ a lightweight method that uses Morton codes to reorder points into a regular pattern. This process is applied to centroids, grouping them into multiple regions, with each region exhibiting distinct feature levels. To optimize the feature computation of each region, we first decompose features into a base vector and an offset tensor. Owing to the similarity in features, the offset tensor falls within a narrow range, providing opportunities to quantize them into low precision. As a result, the feature computation can be separated into two concurrent paths at the top level, one for the high-precision convolution of the base and the other for the low-precision convolution of the offset tensor. Moreover, considering that different regions present varying offset distributions, we propose to apply different precision at the region level, which is also regarded as the second-level quantization in the offset path. As demonstrated in Fig. 1, given two regions, the offset tensor in Region 1 falls within a relatively narrow range, so we quantize them into 4-bit. While the offset tensor in Region 2 exhibits a more dispersed distribution, which requires a wider bitwidth and we employ 8-bit quantization instead.

This work is partly supported by the National Natural Science Foundation of China (Grant No. 62202288). *Zhuoran Song is the corresponding author.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3655905

In summary, the major contributions of this paper include:

- We introduce a Morton-code-based fine-grained quantization algorithm. To the best of our knowledge, we are the first to introduce fine-grained quantization into PCNNs. We utilize Morton codes to capture the spatial locality among centroids and design a two-level mixed-precision quantization at the region level to reduce the computational workloads.
- We propose an accelerator architecture to support our MoC algorithm. We parallel the execution of KNN with Morton code generation and sorting, and employ a conditional Fetch-on-Demand dataflow at the region level to hide latency between aggregation and feature computation.
- We evaluate MoC on various benchmarks, demonstrating that it achieves remarkable performance compared to state-of-the-art PCNN accelerators with satisfactory accuracy.

2 BACKGROUND AND MOTIVATION

2.1 Point Cloud Neural Network

The primary stages of PCNN include FPS, KNN, aggregation, and feature computation. 1) FPS is commonly used to down-sample the point cloud, which exhibits stronger robustness than random sampling. It initially selects a point in random as the first centroid. Then, compute distances between the centroid and all points. During each iteration, the nearest distance for each point is kept, and the point with the maximum distance is selected as the next centroid. This process is repeated until centroids reach the predefined count. 2) KNN is utilized to establish relations among points. It calculates point distances and picks the top-K nearest points as neighbors. The computation of each point is independent, taking advantage of the hardware parallelism. 3) The aggregation stage is set to gather point features. It first subtracts the centroid coordinates from its neighbors. Then, the delta is concatenated with neighbor features, constructing the input local features for the feature computation. 4) The feature computation is typically implemented using Multi-Layer Perceptions (MLPs), which comprises layers of convolution with a kernel size of 1×1, batch normalization (BN), and activation.

2.2 PCNN Accelerator

In an effort to optimize the performance of PCNN, several prior works accelerate it from different aspects. Mesorasi [4] uses an approximation algorithm to exchange the order of the aggregation and feature computation, which applies MLPs on each point coordinates for once. The approach eliminates redundancy introduced by KNN and aggregation, including the replicated computations of centroids and that of the points located at the intersection of adjacent centroids. PRADA [13] presents another approximation algorithm. It utilizes K-means clustering to group centroids and combines them into a single centroid for each cluster. Similar to Mesorasi, the point features are calculated only once, filtering the repeated features and restoring them through a direct copy. PointAcc [8] unifies FPS and KNN on a set of components, which reduces data movement costs between heterogeneous platforms. MARS [17] aims at FPS, proposing a distance filtering technique that constrains the search scope of FPS. It also introduces a scalable and re-configurable elastic array to improve the utilization of the large-scale computing unit.

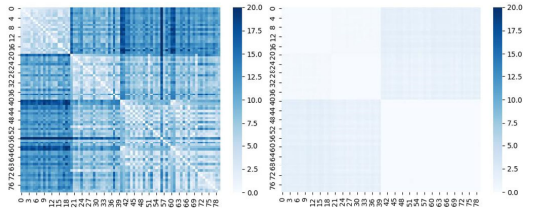

Figure 2: Feature similarity of (a) unordered points, and (b) points in spatial orders using Morton-code-based sorting.

2.3 Neural Network Quantization

Quantization is commonly used to compress the input features or parameters of neural networks. It can be roughly categorized into fixed-length quantization and mixed-precision quantization. The fixed-length quantization is further divided into uniform and non-uniform types. In terms of a n-bit uniform scheme, it maps the data uniformly to 2^n quantization levels. For an input value x, the quantization is performed as Eqn. 1-3, where the function $h^{-1}(...)$ clamps the quantized values into the range of $[-2^{b-1}, 2^{b-1}-1]$, r represents the original real number and q denotes the quantized values. However, the uniform intervals may not suit the varying data distributions, and therefore, the non-uniform quantization is applied. For instance, to fit the Gaussian-like distribution, APoT [6] quantizes data as a sum of power-of-two values, which is non-uniformly distributed along the real number field. Given that different layers in Deep Neural Network (DNN) exhibit various importance and sensitiveness, being vulnerable to the bitwidth of data, mixed-precision quantization is introduced. For example, BitFusion [12] adopts a layer-wise mixed-precision quantization method and implements the algorithm on a reconfigurable MAC array.

$$Q(x) = h^{-1}(round(\frac{x}{scale}) + zeroPt) \tag{1}$$

$$scale = \frac{r_{max} - r_{min}}{q_{max} - q_{min}} \tag{2}$$

$$zeroPt = round(q_{max} - \frac{r_{max}}{scale}) \tag{3}$$

2.4 Motivation

Some studies like PRADA and EdgePC [16] have recognized the correlation between points' spatial locations and features. They use clustering or KNN to capture such locality. However, both existing methods are suboptimal: 1) Clustering involves substantial expensive iterations before convergence. 2) Although KNN is inherent to PCNNs, making it easier to obtain the spatial information than clustering, we observe that points with similar features surpass the scope of neighborhoods created by KNN (refer to Fig. 2, where the similar points are 4× greater than that of the neighbors). To further reduce computations, we intend to capture the point spatial locality at a larger scale using a lightweight method.

Morton code [1] is suitable for establishing relations among points. It is a space-filling Z-curve that maps multidimensional data to one dimension while preserving points' locality. After sorting points based on Morton codes, the point cloud becomes regular, where the unordered points are reorganized into spatially ordered ones. To validate the effectiveness of Morton-code-based sorting in gathering points with similar features, we present a pair

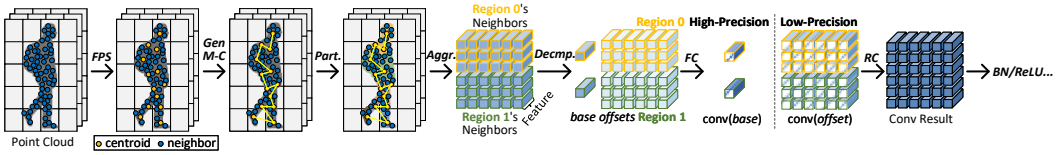

Figure 3: An overview of our MoC Algorithm.

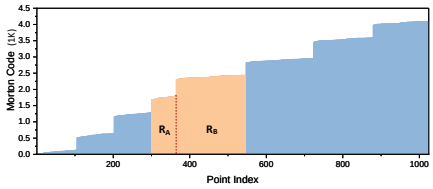

Figure 4: Morton codes exhibit a stepwise distribution.

of heatmaps in Fig. 2, illustrating the similarity of features between unordered points and spatially ordered points after Morton-code-based sorting. The similarity is quantified using L2 distance between features, where lighter colors represent closer distances, indicating higher similarity. It is observed that the color becomes considerably lighter after applying Morton-code-based sorting, demonstrating that points with similar features are properly gathered.

3 ALGORITHM

In this section, we introduce our Morton-code-based fine-grained quantization (MoC) algorithm from two perspectives. First, we clarify the Morton-code-based blocking strategy, which is the foundation of fine-grained. Then, we illustrate a two-level mixed-precision quantization method to compress the bitwidth of features at the region level, which facilitates the computation reduction for PCNNs.

3.1 Morton-Code-Based Blocking Strategy

Morton codes for point coordinates can be generated through coordinate transformation [16] and bit interleaving. In particular, we start by converting the floating-point coordinates into integers. This is achieved by subtracting the origin coordinates and dividing by the resolution, which obtains an integer index for the point in the new coordinate system. Then, we convert the decimal values to binary and interleave bits of each coordinate. The interleaving process follows the pattern "$zyxzyx...zyx$", where x, y, and z represent bits from the 3D coordinates, respectively. For example, the point with transformed coordinates of $(8, 12, 9) = (1000, 1010, 1001)_2$ can be encoded as a Morton code of $(111000010100)_2 = (3604)_{10}$. Notably, the Morton codes generation for different points is independent, suggesting that it can be accelerated through hardware parallelism.

After acquiring Morton codes, we utilize them to sort points in spatial order. We tend to gather nearby points with similar features into a region. Fig. 4 presents Morton codes of a reordered point cloud in ModelNet40 [14], showcasing a stepwise distribution. A natural partitioning method involves segmenting points along the margins of steps. However, this results in significant variation in point numbers across different regions. For instance, in Fig. 4, the two adjacent regions R_A and R_B exhibit a noticeable difference in point numbers. Such variation in region size is unfriendly to hardware, which is not conducive to batch processing. To tackle this problem, we partition the regions at a fixed interval, with each region size constrained to a consistent value.

3.2 Two-Level Mixed-Precision Quantization

Owing to the similarity of features among nearby points, we incorporate a feature decomposition stage that breaks down features of each region into a base vector and an offset tensor, with offsets falling within a narrow range. The base vector is specifically composed of means of features, and the offset tensor is generated by subtracting the base from features, whose shape is consistent with the features'. To alleviate the workload and memory overhead of feature computation, we propose a two-level mixed-precision quantization method. At the first level, we quantize offsets to lower precision, while preserving bitwidth of the base to ensure accuracy.

Given that different regions exhibit diverse levels of importance, some regions show high variance in offset values, while others show similar values. Therefore, at the second level, we apply mixed-precision quantization for the offsets across regions to leverage different data distributions. To determine the quantization precision for each region, a straightforward practice involves producing a histogram of the data distribution and then selecting the bitwidth based on the shape of the curve. A peaked distribution in the curve suggests a narrow data range that can be represented with a short bitwidth. However, computing histograms for all regions is costly. To simplify the computation, we use confidence probability (as shown in Eqn. 4, where r is the offset) to approximately detect the offset distribution. The higher the confidence probability, the more offsets clustered around the mean, signifying that the curve tends to a peak. Consequently, a lower quantization precision can be applied. We use two types of low-precision, including 4-bit and 8-bit, to quantize offsets in different regions. The decision is made by comparing the confidence probability with a threshold T. When $\gamma \geq T$, we use 4-bit quantization; conversely, 8-bit is applied.

$$P(Mean - \alpha(Max - Min) \leq \boldsymbol{r} \leq Mean + \alpha(Max - Min)) = \gamma \quad (4)$$

In alignment with feature decomposition, the feature computation is departed into two paths, one for the base and the other for the offset tensor. The base path involves a direct convolution for the base vector, whereas the offset path incorporates an extra dequantization operation after calculating the convolution results for the low-precision offset tensor. Finally, to reconstruct the feature convolution results for each point, the two paths' outcomes are combined through an element-wise summation.

3.3 Walk-Through Example

We demonstrate the whole process of our MoC algorithm in Fig. 3. The Morton code is generated (*abbr. Gen M-C*) for each centroid set, which is then utilized to reorganize points in spatial orders. The regularized points are further grouped into multiple regions, with each region exhibiting strong similarity in features. The aggregation stage takes information from both KNN and partition to gather features at the granularity of regions. Then, the feature decomposition is performed, producing a set of base vectors and a bunch

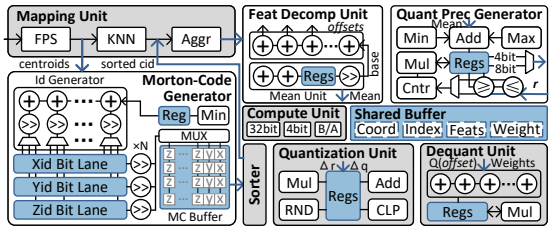

Figure 5: An overview of the MoC architecture.

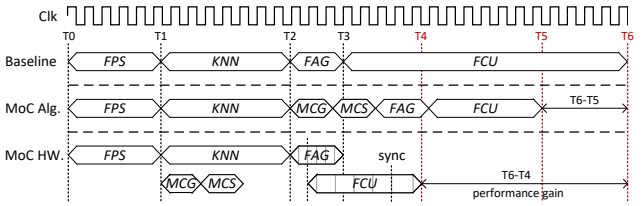

Figure 6: Compare time sequence of the MoC (Alg. and HW.).

of offset tensors. The convolution operations are separated into a high-precision path for bases and a low-precision path for offsets. Eventually, the results from two paths are summarized element-wise to reconstruct (*abbr. RC*) the feature convolution result.

3.4 Design Exploration

In our MoC algorithm, there are two parameters left to explore. The threshold T, which determines the ratio of 4-bit and 8-bit, involves a trade-off between performance and accuracy. A lower threshold introduces more 4-bit regions, significantly reducing the feature computation workloads. However, this results in unacceptable accuracy loss, as multiple similar offsets are replaced by a single value. Consequently, selecting an optimal threshold T is essential. Regarding the region count, which also impacts both performance and accuracy, a larger number of regions may lead to fewer 4-bit offsets due to the uniform distribution within a narrow region, thereby diminishing the benefits of our MoC algorithm. Conversely, as the region count decreases to 1, it degrades to the baseline without Morton-code-based sorting, leading to unacceptable accuracy loss. Therefore, we should carefully determine the region count.

4 ARCHITECTURE

To support our MoC algorithm, we co-design an accelerator architecture. As depicted in Fig. 5, the entire system comprises: the mapping unit implements FPS, KNN, and aggregation, utilizing a well-established design from PointAcc. The Morton-code generator and quantization precision generator are key elements in our design, and we elaborate on them in detail below. The feature decomposition unit separates features into base and offsets. The quantization and dequantization units take responsibility for compressing the offsets and recovering the low-bit convolution results to full precision, respectively. The computing unit (CU) incorporates a 32-bit Multiply-Accumulate (MAC) array for base convolution and feature reconstruction, a 4-bit MAC array for convolving quantized offsets, and a batch normalization/activation (*B/A*) unit. The shared buffer holds point coordinates, indices, features, and weight matrices.

4.1 Morton-Code Generator

The Morton-code generator calculates Morton codes for centroids, receiving point coordinates and producing Morton codes for the succeeding sorter. It begins by traversing point coordinates to find the absolute minimum for each dimension. The minimum is then subtracted from point coordinates parallelly in the id generator. Then, the differences are divided by the resolution to produce integer indices in the new coordinate system. To improve efficiency, we set the resolution as a power of 2, implementing division using shifters. Integer indices are buffered in individual *xid* (*yid* or *zid*) bit lanes. Bitwise interleaving is achieved using shifters attached to each bit lane, picking the least significant bit from the binary coordinate per cycle and sending it to the multiplexer (*abbr. MUX*) to further determine the address stored in the MC Buffer. Benefiting from 3-way parallel processing, the latency of this stage is shortened. Moreover, this component operates in parallel with the KNN component, hiding the cost of Morton code operations.

4.2 Quantization Precision Generator

The quantization precision generator determines the bitwidth for the offset tensor. As illustrated in Eqn. 4, the confidence interval ranges from $Mean - \Delta(\text{offsets})$ to $Mean + \Delta(\text{offsets})$, where $\Delta(\text{offsets}) = Max - Min$. To implement this process, it initially acquires the mean of the offset tensor, and calculates $\Delta(\text{offsets})$ using the maximum and minimum values of offsets. The scaled $\Delta(\text{offsets})$ generated by the multiplier is then summarized with the mean to form the confidence interval. Afterward, we calculate the probability that offsets fall within the confidence interval. This is achieved by a pair of comparators, a counter, and a multiplier. The confidence probability is then delivered to the comparator to compare with the threshold T, determining the bitwidth for quantization. Notably, the mean unit, embedded in the feature decomposition unit, can be reused to calculate the mean of the offset tensor.

4.3 Dataflow

In the context of our Morton-code-based blocking strategy, the gathered regions arrive asynchronously, and high-dimensional features contribute to long memory access latency, resulting in early-arriving regions waiting for the late ones. This drawback is inherent to the traditional Gather-MatMul-Scatter dataflow, requiring all regions to be ready before starting the feature computation. To address this concern, we employ a conditional Fetch-on-Demand (FoD) dataflow at the region level, which starts the feature computation once a region is gathered. Therefore, memory access latency is hidden by early regions' feature computation. Furthermore, to ensure correctness, we set a synchronization barrier before BN, which applies to all regions, serving as the condition that constrains FoD.

To demonstrate the necessity and superiority of our proposal, we present timelines comparing our MoC algorithm and architecture with the baseline, in Fig. 6. The primary benefit of the MoC algorithm is achieved through low-precision convolution. However, considering a worst-case where different stages are implemented in separate kernels on GPU, the KNN and Morton code generation are serialized, which hampers performance. To eliminate this concern, we design a customized hardware architecture that parallels the Morton code generation and sorting with KNN, properly overlapping their execution latency. We also shorten the time duration of

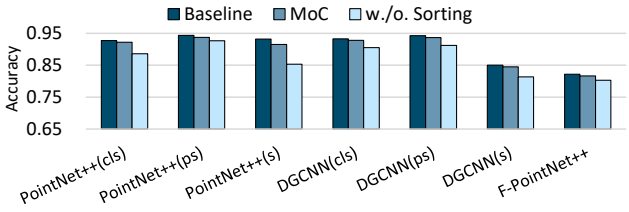

Figure 7: The accuracy of MoC algorithm compared with the baseline and an ablation variant.

the Morton code generation by utilizing the specialized Morton-code generator. Additionally, the conditional FoD dataflow at the region level contributes significantly to performance gains by hiding the memory access latency in aggregation. All these factors contribute to the outstanding performance of MoC.

5 EVALUATION

5.1 Evaluation Setup

Benchmarks. We pick 7 representative PCNNs as benchmarks. These compress the classification, part segmentation, and semantic segmentation networks of PointNet++ and DGCNN [10], along with the detection network of F-PoitnNet++ [11]. These PCNNs exhibit different structures and operate on extensive datasets, including ModelNet40, ShapeNet [3], S3DIS [2] and KITTI [5], each with various point scales ranging from 1K to 4K. All these benchmarks contribute to a comprehensive evaluation of our proposal.

Architectural Modeling. To model the behavior of our MoC architecture, calculating cycles and emulating read/write operations on SRAMs, we develop a cycle-accurate simulator. To provide reliable DRAM access latency, we integrate DRAMSim3 [7], with a configuration of 8GB HBM2. The on-chip buffers are estimated through CACTI [9] under 40nm technology node.

The hardware configurations are detailed in Table 1. We compared the MoC architecture with several prior works, including PointAcc, MARS and PRADA. In terms of the on-chip memory system, we set up a total of 450KB SRAMs, with the shared buffer contributing the most, occupying 429.5KB. Given that we quantize a part of features into low-precision and the conditional FoD dataflow does not force all features being held on-chip, the memory footprint of features is significantly reduced compared to PointAcc and MARS, as illustrated in Table 1. Regarding the CU, we employ a 4×64 32-bit MAC array for base convolution and result reconstruction, where the array supports up to 4 concurrent base vectors. Additionally, an $8 \times 16 \times 128$ 4-bit MAC array is employed for the offset tensor convolution, which is designed to synchronize execution latency with the 32-bit array. Notably, since the area of a 32-bit MAC unit is approximately 64× larger than a 4-bit MAC unit in TSMC 40nm technology library [15], we can equate (EQV) these two arrays to a 32-bit array with 512 MACs. Our MoC architecture achieves outstanding peak performance, outperforming PointAcc and MARS by 2.1×, with only 12.5% EQV MACs of theirs.

Baselines. We compare our MoC architecture with 3 types of hardware platforms, including CPU (Intel Xeon Gold 6226R), GPU (Server GPU: NVIDIA A100 80GB PCIe, Edge GPU: Jetson AGX Xavier), and state-of-the-art ASICs (PointAcc, MARS and PRADA). For a fair comparison, we align all platforms to an equivalent number of MAC units by scaling their performance.

Table 1: Hardware Configurations

Configuration	PointAcc	MARS	PRADA	**MoC**
MACs	32b: 64x64	32b: 64x64	32b: 16x16	32b: 4x64; 4b: 4x16x128
SRAM	776KB	776KB	98KB	450KB
DRAM	HBM2	HBM2	HBM2	HBM2
Bandwidth	256GB/s	256GB/s	256GB/s	256GB/s
Frequency	1GHz	1GHz	1GHz	1GHz
Technology	40nm	40nm	40nm	40nm
Area (mm2)	15.7	-	3.9	7
Peak Performance	8TOPS	8TOPS	2.2TOPS	16.7TOPS

5.2 Accuracy

To verify the impact of the MoC algorithm on inference accuracy, we implement it on GPU using PyTorch. As shown in Fig. 7, our MoC algorithm achieves a fairly close accuracy to the baseline implementation. On average, the accuracy loss of MoC is 0.817%, which is deemed acceptable in the PCNN field. To further validate the effectiveness of Morton-code-based sorting, we design an ablation experiment, referred to as "w./o. sorting". This experiment removes the Morton code generation and sorting from the MoC algorithm, leaving the feature computation directly on the unordered points. Due to the absence of spatial locality among points, the similarity in features within a region becomes fragile. In this context, the distribution of offsets is likely to be loose, resulting in wide quantization intervals to cover the data range and leading to significant accuracy loss. Specifically, the average accuracy loss of "w./o. sorting" is 4.138%, which is completely unacceptable. Therefore, the Morton-code-based sorting is crucial to our MoC algorithm.

5.3 Performance and Energy Efficiency

We present the speedup and energy savings of MoC architecture in Fig. 8 and Fig. 9, respectively. On average, our MoC offers a speedup of 12×, 6.3×, 4.7×, 3.8×, 3.4× and 2.8× over CPU, Server GPU, Edge GPU, PointAcc, PRADA and MARS. In comparison with CPU and GPU, our MoC manifests the benefits of customized architecture, which possesses higher resource utilization and shorter execution latency. Additionally, our MoC outperforms the state-of-the-art PCNN accelerators: 1) PointAcc focuses on reducing data movement in feature computation, which is less critical than heavy computational workloads, especially in the part segmentation network of PointNet++. Therefore, our MoC achieves the most significant performance gains over PointAcc in PointNet++(ps). 2) MARS is committed to optimizing FPS, where the stage is not predominant for most benchmarks on PCNN accelerators, except the semantic segmentation network of PointNet++ and DGCNN. Although MARS improves the utilization of CUs, it is inferior to our method since we further reduce the feature computation workloads by converting a large number of computations into a low-precision mode. 3) PRADA incorporates an additional time-consuming clustering stage to group local pairs, accounting for 46.7% of the overall process. For energy efficiency, our MoC offers an average savings of 19.3×, 9.7×, 6.0×, 5.2×, 4.6× and 4.1× over CPU, Edge, Server GPU, PointAcc, PRADA and MARS. Owing to the mixed-precision quantization algorithm, the volume of data read and write during memory access can be greatly reduced, resulting in substantial energy savings in both memory access and computation.

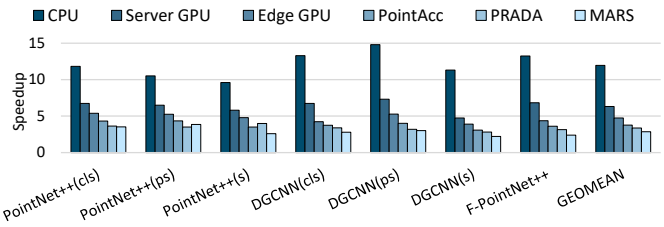

Figure 8: The speedup of MoC over CPU, GPU and ASICs.

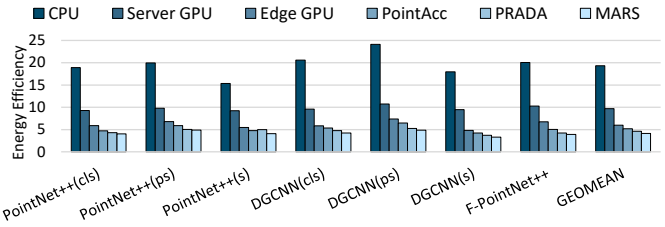

Figure 9: The energy savings of MoC over CPU, GPU and ASICs.

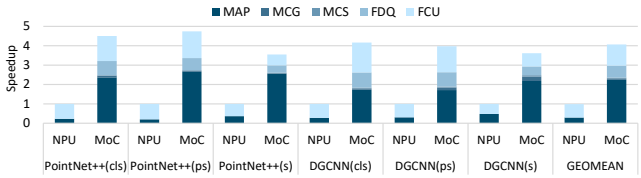

Figure 10: The latency breakdown of each stage in MoC.

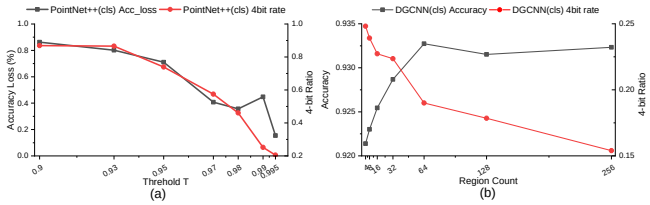

Figure 11: (a) Explore the parameters of (a) the threshold T, and (b) the region count.

5.4 Exploration

Latency Breakdown of Stages. We break down the execution latency of stages in the MoC architecture, as demonstrated in Fig. 10. To emphasize the benefits, we compare the MoC with an NPU developed upon a variant of our architecture. This variant removes all optimizations, leaving only the essential components, such as the mapping unit and a 32-bit CU with 512 MACs, aligning with our MoC architecture, to support PCNNs' execution. It is notable that the feature computation (FCU) dominates the entire process in NPU. While the proportion of this stage is obviously decreased after applying our MoC architecture. On average, it reduces 62.4% of FCU compared to NPU. Additionally, the extra stages we introduced, such as the Morton code generation (MCG), Morton-code-based sorting (MCS), feature decomposition and quantization (FDQ), only contribute a small portion to the whole process.

Parameter Tradeoff. As illustrated in Section 3.4, the threshold T affects both performance and inference accuracy. We conduct an experiment on PointNet++(cls) to investigate the optimal threshold value, as shown in Fig. 11(a). As the threshold T increases from 0.9 to 0.995, the proportion of 4-bit offsets decreases from 83.6% to 0.6%, leading to an accuracy loss ranging from 0.88% to 0.32%. To strike a balance between the two factors, we find that when $T = 0.97$, the model suffers low accuracy loss with 52.6% 4-bit offsets, contributing to a huge EQV. FP32 MACs reduction of 81.2%.

We additionally conducted an experiment on DGCNN(cls) with varying region counts, as depicted in Fig. 11(b). As the region count increases from 4 to 256, the region size becomes smaller, where the offsets exhibit a uniform distribution around the mean. Therefore, seldom regions are quantized into 4-bit, resulting in a decreasing trend of the 4-bit occupancy. To preserve accuracy while achieving optimal performance, we select the region count corresponding to the intersection point of the two curves, which is 32 in this context.

6 CONCLUSION

This paper introduces fine-grained mixed-precision quantization into PCNNs for the first time. We propose an algorithm-architecture co-designed scheme named MoC, which is developed on the similarity of nearby points' features. To capture their spatial locality, we utilize Morton-code-based sorting and partition them into multiple regions. To reduce feature computation workloads, we decompose features into base and offsets, with offsets falling within a narrow range. We further propose a two-level mixed-precision quantization and perform low-precision convolutions. The final results are reconstructed using both convolution results from the base and offsets. For the hardware design, we parallel Morton-code-related operations with KNN. Additionally, a conditional FoD dataflow is introduced to support asynchronous computation at the region level. The MoC is evaluated in extensive experiments, demonstrating outstanding performance and satisfactory accuracy.

REFERENCES

[1] Jeroen Baert. 2013. Morton encoding/decoding through bit interleaving: Implementations. *retrieved from, Oct* 7 (2013), 13.
[2] Armeni et al. 2016. 3d semantic parsing of large-scale indoor spaces. In *CVPR*. 1534–1543.
[3] Chang et al. 2015. Shapenet: An information-rich 3d model repository. *arXiv:1512.03012* (2015).
[4] Feng et al. 2020. Mesorasi: Architecture support for point cloud analytics via delayed-aggregation. In *MICRO*. IEEE, 1037–1050.
[5] Geiger et al. 2012. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*. IEEE, 3354–3361.
[6] Li et al. 2019. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. *arXiv:1909.13144* (2019).
[7] Li et al. 2020. DRAMsim3: A cycle-accurate, thermal-capable DRAM simulator. *IEEE COMPUT ARCHIT L* 19, 2 (2020), 106–109.
[8] Lin et al. 2021. Pointacc: Efficient point cloud accelerator. In *MICRO*. 449–461.
[9] Muralimanohar et al. 2009. CACTI 6.0: A tool to model large caches. *HP laboratories* 27 (2009), 28.
[10] Phan et al. 2018. Dgcnn: A convolutional neural network over large-scale labeled graphs. *NN* 108 (2018), 533–543.
[11] Qi et al. 2018. Frustum pointnets for 3d object detection from rgb-d data. In *CVPR*. 918–927.
[12] Sharma et al. 2018. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *ISCA*. IEEE, 764–775.
[13] Song et al. 2023. PRADA: Point Cloud Recognition Acceleration via Dynamic Approximation. In *DATE*. IEEE, 1–6.
[14] Wu et al. 2015. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*. 1912–1920.
[15] Yang et al. 2022. DTQAtten: Leveraging dynamic token-based quantization for efficient attention architecture. In *DATE*. IEEE, 700–705.
[16] Ying et al. 2023. EdgePC: Efficient Deep Learning Analytics for Point Clouds on Edge Devices. In *ISCA*. 1–14.
[17] Yang et al. 2023. An Efficient Accelerator for Point-based and Voxel-based Point Cloud Neural Networks. In *DAC*. IEEE, 1–6.