Sava: A Spatial- and Value-Aware Accelerator for Point Cloud Transformer

Xueyuan Liu[1], Zhuoran Song[1]* Xiang Liao[1], Xing Li[1], Tao Yang[2], Fangxin Liu[1] and Xiaoyao Liang[1]
[1]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[2]Huawei Technology Co., Ltd.
[1]{xueyuan_liu, songzhuoran}@sjtu.edu.cn

***Abstract*—Point Cloud Transformer is undergoing a rising trend in both industry and academia. It aligns traditional point cloud feature extraction methods with the latest transformer architecture and achieves remarkable performance. However, accelerators for traditional point cloud neural networks (PCNNs) and those solely for transformers fail to capture the characteristics of point cloud transformers, thus exhibiting poor performance. To address this challenge, we propose Sava, a co-designed accelerator that adopts a spatial- and value-aware hybrid pruning strategy for point cloud transformers. In terms of the spatial domain, we observe that points in regions of various densities exhibit different levels of importance. In the value space, a minor input contributes less to features, indicating lower importance. Considering both perspectives, we hybridize the information inherited from the spatial and value spaces to prune less significant values in attention, which converts data to sparse patterns and makes it readily accelerated. Furthermore, we adopt low-bit quantization to boost computations and apply varying quantization precisions across different network layers based on their sensitivity. In support of our algorithm, we propose an architecture that employs a configurable mixed-precision systolic array for various computing loads under diverse precisions. To address the workload imbalance of the unstructured sparse computations, we introduce a data rearrangement mechanism, which improves resource utilization while hiding latency. We evaluate our Sava on four point cloud transformer models and achieve notable accuracy and performance gains. In comparison with CPU, GPUs, and ASICs, our Sava offers** $10.3\times$**,** $3.6\times$**,** $3.3\times$**,** $2.6\times$**,** $2.2\times$ **speedup, along with** $20\times$**,** $8.8\times$**,** $6.9\times$**,** $3.2\times$**,** $2.4\times$ **energy savings on average.**

I. Introduction

Most of point cloud transformers can be categorized as a hybrid of traditional PCNNs and transformers. As depicted in Fig. 2, they incorporate traditional PCNNs as the front-end and transformers as the back-end. Owing to their higher accuracy than traditional PCNNs, they play a vital role in autonomous driving, Augmented Reality (AR), and so on. The point cloud transformer takes benefits of both the ability to capture local features in PCNNs and the capability to model the global context in transformers. However, the increasing point scale and the complexity of stages make it far from real-time processing.

Several prior PCNN accelerators only focus on the front-end of the point cloud transformer, resulting in poor performance across the entire network due to varying bottlenecks (see Fig. 1(a) where some models are hindered by the back-end) and their lack of support for the back-end transformers. For instance, Mesorasi [1] proposes delayed aggregation to alter the execution order of the feature computation and aggregation

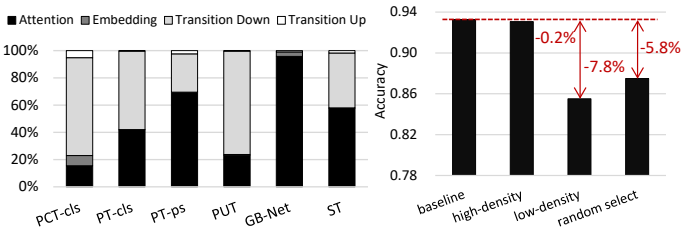

Fig. 1. (a) Latency breakdown of stages in point cloud transformers. (b) The regions with different densities exhibit various importance.

stages in the front-end, significantly reducing the workload of feature computation. However, Mesorasi fails to optimize point cloud transformers when the back-end transformers dominate the performance, as in PT-ps [2] and GB-Net [3]. PRADA [4] unveils the redundancy that exists in the feature computation of similar vectors, and utilizes a dynamic approximation algorithm to alleviate their computations. But it introduces substantial approximation within multilayer perceptions (MLPs), resulting in unacceptable accuracy loss when the loss is propagating and accumulating with increasing layers of the back-end transformers. In a word, all these PCNN accelerators are not specially designed for point cloud transformers as they ignore the connections between the front-end and the back-end, hampering their adaptability in the point cloud transformers.

To tackle this dilemma, we consider both the front- and back-end of the point cloud transformer, introducing a spatial- and value-aware hybrid pruning strategy to the attention mechanism. This approach reduces redundant computations and creates opportunities for further optimizations for sparse patterns. In the spatial domain, we observe that different regions in point clouds exhibit various densities, indicating different importance. This information can be utilized to guide pruning. To implement it, we generate a spatial-aware attention mask from score vectors, which is a representation of the regional importance. In the value space, we distinguish importance based on the contribution of values in attention. Similarly, we generate a value-aware attention mask to assist pruning. A single pruning strategy based on the specific domain cannot capture the comprehensive properties of the point cloud, which may adversely affect accuracy. Therefore, we hybridize both pruning strategies to form a hybrid mask and apply it to the attention mechanism, converting dense patterns to sparse modes and providing opportunities for hardware acceleration.

To further capture the benefits of the hybrid pruning strategy, we propose Sava architecture. It utilizes an extended KNN

* Zhuoran Song is the corresponding author.

979-8-3503-4859-0/DATE24/© 2024 EDAA

module to generate score vectors. Given that the feature computation is performed with high-precision to ensure accuracy, while the attention mask is calculated with low-precision for a fast generation, we employ a reconfigurable mixed-precision systolic array to accommodate different bitwidths of data. After applying the hybrid mask generated from the systolic array to prune attention, we obtain sparse patterns. We skip the multiplications of those less important values by decoupling the compute cells and registers of the systolic array and managing the computations flexibly. To address workload imbalance in unstructured sparse computations, we introduce a mechanism that packs rows with similar sparsity for execution, thereby maximizing resource utilization while hiding latency.

In summary, we make the following contributions:

- **Hybrid Pruning Strategy:** We leverage the spatial information derived from the front-end and value importance implied from the back-end to construct a hybrid pruning strategy, which helps to reduce redundant computations without introducing obvious accuracy loss.
- **Sava Architecture:** In support of our algorithms, we propose Sava architecture. To the best of our knowledge, Sava is the first accelerator for the point cloud transformer. It incorporates a reconfigurable mixed-precision systolic array and adopts a data rearrangement mechanism.
- **Evaluation:** Extensive experiments show that Sava achieves remarkable accuracy and performance than the state-of-the-art PCNN accelerators.

II. Background and Related Work

A. Point Cloud Transformer

The point cloud transformer is built upon traditional PCNNs, which primarily introduces transformers to replace the feature extraction network (*e.g.* MLP) or incorporates them as the back-end. Fig. 2 illustrates the basic structure of the point cloud transformer. It comprises the transition-down, embedding and transformer modules. The transition-down is derived from PointNet++ [5], including the farthest point sampling (FPS), neighbor search and grouping. some variants may also employ MLPs to extract local features of points. The embedding is a pre-stage of the transformer, converting the output features of the transition-down into the input format required for the transformer. The transformer consists of multiple attention layers, which is utilized to extract features. The key operators in the point cloud transformer are described in detail:

*1) **FPS**:* FPS is a widely used down-sampling algorithm in PCNNs. It takes point coordinates as inputs, randomly selects the initial centroid, and iteratively generates new centroids by searching for the farthest point from the current centroid set.

*2) **KNN**:* K-Nearest-Neighbor (KNN) is a widely-used neighbor search algorithm to determine the point neighborhood. It calculates distances between the centroid and all points in each iteration, selecting the K nearest points as neighbors.

*3) **Attention**:* Attention is a basic building block of the transformer, it converts the input into query (Q), key (K) and value (V) matrics through linear operations or simple transformation. Then, QK^T obtains the attention score (S) matrix, followed by activation and multiplication with the matrix V. Those results are further transformed using some post-processing components like linear or MLPs.

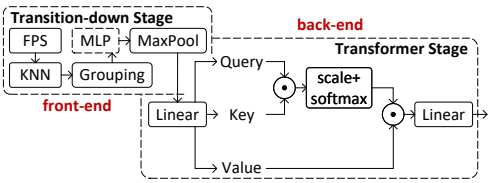

Fig. 2. A paradigm of the point cloud transformer.

B. Related Work

Numerous previous works are committed to optimizing the front-end of the point cloud transformer. For instance, Mesorasi [1] proposes delayed aggregation to postpone the aggregation behind feature computation, which helps to reduce the workload of MLPs but deteriorates the feature access latency. PointAcc [6] implements transition-down operations in Application-Specific Integrated Circuit (ASIC) and proposes a configurable memory management unit, coupled with a fetch-on-demand computation flow, to maximize data reuse. However, PointAcc lacks support for transformer dataflows, leading to suboptimal performance on modern transformer-based PCNNs. PRADA [4] introduces dynamic approximation to feature computation, which reduces redundant accesses and computations for similar local pairs. Notably, it incorporates an extra clustering stage, creating new data dependencies along the critical path, resulting in unexpected costs. Furthermore, PRADA demonstrates its effectiveness only when the relative coordinates between centroids and their neighbors are concatenated with centroid features, suggesting that the centroid features dominate feature vectors. This condition is limited across diverse PCNNs. Notably, all these methods give no support for the latest point cloud transformers.

III. Algorithm Design

In this section, we introduce a spatial- and value-aware hybrid pruning strategy, which prunes uncritical values of the attention score matrix S and skips redundant computations in both QK^T and SV in the attention mechanism. This approach captures spatial information and identifies uncritical regions in the point cloud. Additionally, it finds values with different importance in the matrix S. Considering the properties of the point cloud transformer, the hybrid pruning strategy reduces substantial redundant computations and memory costs.

A. A Spatial-Aware Pruning Strategy

To explore the potential for accelerating point cloud transformers, we generate a mask from the transition-down to guide pruning for transformers. Due to the non-uniform and clustered point distribution, different regions exhibit varying densities. Intuitively, points located in diverse regions present different importance. The regions with higher density contain much more redundant information, thus leading to a negligible performance loss when pruning a portion of points. On the contrary, the points within lower density regions are more critical, which

profoundly impact accuracy when removing some of them. Fig. 1(b) validates our assumption in practice. We prune 30% points within high-density, low-density and randomly selected regions, finding that the minimum accuracy loss is obtained from high-density regions, which is aligned with our assumption.

To represent the density of a region, we employ the average distance between the centroid and its neighbors. The equation is shown as Eqn. 1, where d^c_{avg} denotes the average distance between the centroid c and its neighbors, K indicates the number of neighbors, P_c and P_i represent the coordinates of the centroid c along with its neighbor i, respectively.

$$d^c_{avg} = \frac{1}{K}\sum_{i=1}^{K} \|P_c - P_i\|_2^2 \tag{1}$$

To inherit the spatial features to transformer blocks and match the shape of S in attention, we construct a **spatial-aware attention mask** (*abbr.* $smask$) to direct the pruning for S. Specifically, we first generate a Q-score vector by ranking the average distance from 0 to $N-1$, where N is the point number in a point cloud. The minor the score, the smaller the average distance, implying a lower importance of the centroid. Then, we transpose the Q-score and obtain a K-score vector, similar to the attention mechanism. We multiply Q-score and K-score vectors to form a score matrix with the same shape as S (see Eqn. 2). Afterward, we leverage a spatial pruning ratio to binarize the score matrix. Specifically, we set small scores to 0 while keeping the large ones by setting them to 1. Up till now, we have obtained a pruning mask based on the spatial properties of point clouds.

B. A Value-Aware Pruning Strategy

In the value space, we notice that small values in attention contribute less to the output features, indicating negligible importance in feature computation. We create a side path before QK^T and construct a **value-aware attention mask** (*abbr.* $vmask$) to guide pruning. Within the side path, we quantize matrices Q and K into low precision (*i.e.*, 4-bit), allowing a rapid execution without hindering the critical path's performance. Next, we perform General Matrix Multiplication (GEMM) between $quant(Q)$ and $quant(K)$ along with softmax operations similar to that in the attention mechanism (see Eqn. 3). Eventually, the $vmask$ is generated by binarizing the above results based on a specific threshold, which filters uncritical values in the matrix S.

$$smask = binarize(Q_{score} \cdot K_{score}) \tag{2}$$

$$vmask = binarize(softmax(quant(Q) \cdot quant(K))) \tag{3}$$

$$hmask = binarize(\alpha * smask + \beta * vmask) \tag{4}$$

We further explore the potential of computation reduction of $vmask$. Given that quantization is only applied to the side path, which indirectly affects attention through the mask, leading to a pretty minor impact on accuracy. Therefore, we can use variable precisions in quantization across different layers. Specifically, we utilize lower-bit in relatively robust layers to reduce computation and storage, while using higher-bit in sensitive layers to ensure accuracy.

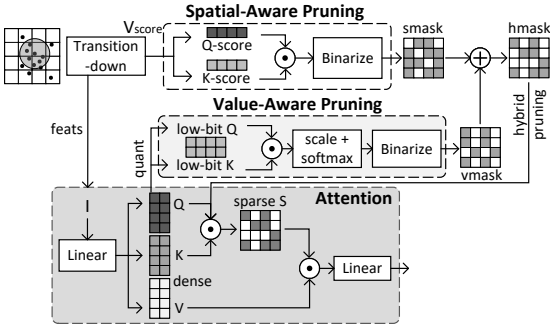

Fig. 3. An overview of the spatial- and value-aware hybrid pruning algorithm.

C. Hybrid Pruning Strategy

Both pruning strategies produce attention masks under the same shape as S. We hybridize them by fusing $smask$ and $vmask$ together with different weights (*i.e.* α and β), as illustrated in Eqn. 4. The weights are determined by a set of empirical values that reflect a declined impact of $smask$ and a rising importance of $vmask$ considering the dependencies of stacking layers.

After generating the hybrid mask (*abbr.* $hmask$) as Fig. 3 shows, we prune redundant multiplications between Q and K in which elements in the $hmask$ are 0. As a result, the matrix S becomes sparse, and SV can be further accelerated by the hardware designed for sparse-dense matrix multiplication (SpMM), which is elaborated in the following section.

IV. Sava Architecture

To support our algorithms and investigate the potential for accelerating sparse patterns, we co-design an architecture named **Sava**. Fig. 4 shows an overview of Sava. It consists of two main components: a transition-down unit and a tensor unit. The transition-down unit handles operations in the transition-down stage, including FPS, KNN, and grouping. Practically, the extended KNN module takes responsibility for KNN and the creation of score vectors. The tensor unit is designed for feature computation and the generation of attention masks. To adapt to various quantization precisions, we propose a mixed-precision systolic array. All details are elaborated in the following subsections.

A. Transition-Down Unit

The spatial-aware attention mask is generated through two rounds of sortings. The first one sorts the average distances to rank the importance of centroids. Then, the second one takes the sorting indices to resume their locations and assigns scores for centroids. We implement this extended KNN module through an insertion-sort array, as depicted in Fig. 4(b), which takes point indices and distances as inputs and produces the sorted indices as outputs. The insertion-sort array comprises a line of sorting elements (SEs). In terms of the KNN execution flow, each SE takes distances generated by the distance calculator (DC) lying on the left side of the array and compares with the current distance held in its register to determine the data

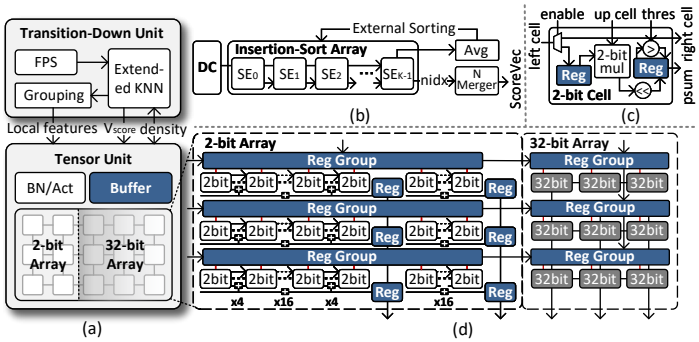

Fig. 4. An overview of the Sava architecture.

movement. If the income distance is smaller than its registered value, the register will be updated and the original value will be shifted to right cells one after another. Otherwise, the distance passes through SE until finding a smaller one to replace. The insertion-sort array keeps the K nearest neighbors of a centroid after all distances are generated in 1 iteration.

We extend the insertion-sort array by introducing an additional output path, providing distances to calculate their average. After generating the average distances for all centroids through an average unit, we separate these distances into blocks of size K, and send them to the insertion-sort array blockwise for the 1st round of sorting. We replicate the sorting for the indices produced by the 1st round to obtain final scores of centroids. Due to the limited length of the insertion-sort array, which is often far smaller than the number of centroids, we adopt external sorting to rank the entire sequence of these average distances. In other words, we still utilize insertion-sort within each block and ultimately merge all the sorted subsequences together via an N merger component.

B. Tensor Unit

All the computations of vectors and matrices are conducted on the tensor unit. It comprises a mixed-precision systolic array, an on-chip buffer and a batch normalization/activation unit. The mixed-precision systolic array is designed for various bitwidths of inputs. For example, MLPs in the transition-down stage and attentions in the transformer stage require 32-bit to ensure accuracy. While the computations of the pruning strategies only require 2-bit or 8-bit for fast processing. If we place all types of data into a full-precision unit, aligning with the highest precision among them, it may lead to a significantly low resource utilization. Thus, we employ a matrix of 2-bit cells to accommodate these low-precision computations.

Given that the multiplications with power-of-2 bitwidths can be decomposed into multiple 2-bit multiplies, which are then combined using a series of adders and shifters. For instance, the 8-bit multiplication can be implemented by combining 16 2-bit multiplies, where the shift steps vary across 0 to 14 bits. The architecture of a 2-bit cell is depicted in Fig. 4(c). Its left input is controlled by an enable signal, which allows data to flow into the register only when the signal is set to 1. Otherwise, the data will pass directly to the following cells, restricting the transfer latency to almost that of the 32-bit array. A shifter is attached to each 2-bit cell, where the step is configured based on different precision modes. The computational result is held in a register. Furthermore, to support the binarize operation in place, we set a comparator next to the result register.

In support of skipping redundant computations indicated by $hmask$, we embed a decoupled register group along each row of cells. The register group accepts both Q and K vectors and sends them to any cell within the row. Therefore, the multiplications are determined by the routing between cells and registers, which is actually controlled by the values of $hmask$. Assume the j-th element in the i-th row of $hmask$ is 1, indicating the multiplication between the i-th row of Q and the j-th row of K is enabled. In this condition, the j-th cell of the i-th line accesses the register holding the j-th row of K. After generating the unstructured sparse matrix S, we keep its non-zero elements stationary on the array, and stream the vectors V from the top to each row's register group. The number of registers is actually larger than that of cells. Therefore, we load a vector of V and only deliver the data to cells when the corresponding locations within S is non-zero. In other words, the routing of elements within V is governed by the indices of non-zero elements of S.

To accommodate higher precisions' computations, we extend the 2-bit array as a supplement to the 32-bit array, especially when the resources are limited in MLPs or transformers. We utilize 32 2-bit cells to calculate the partial sums iteratively and accumulate them through shifters within the cells and adders at the bottom line of cells. The 32-bit result is held in the register at the rightmost of each row. Furthermore, We connect registers vertically and set up an adder at their end to accumulate the partial sums of 32-bit tensor multiplications.

All the required precisions include 2-bit$\times$2-bit, 8-bit$\times$8-bit in attention masks' generation and 32-bit$\times$32-bit in MLPs and transformers. We elaborate on their data flow in this paragraph. The 2-bit$\times$2-bit simply keeps the quantified Q vector stationary on the row and streams the quantified K vectors from the left side of the array. The multiplication is performed just in one cell and the results are exported to the rightmost registers. For the multiplications of 8-bit$\times$8-bit, we utilize 16 2-bit cells in a row to form an execution group, and accumulate partial sums through an adder set at the bottom of the vertically-connected group registers. Regarding 32-bit$\times$32-bit, it involves 256 2-bit multiplications. we implement it by leveraging 32 2-bit cells in a time-division multiplexing mode to calculate partial sums, which are then repeatedly accumulated in each row to obtain the final result. To further keep the data arrangement synchronized with the 32-bit array, we treat each row of the 2-bit array as equivalent to a 32-bit cell, and the $N \times 32$ 2-bit array as equivalent to a column of N 32-bit cells.

Unfortunately, the unstructured sparsity may lead to a load imbalance among different rows. We carry out an experiment on PCT to count the number of non-zero elements in each row of the attention mask. As Fig. 5(a) shows, the distribution of non-zero elements varies across different rows significantly, where the maximum difference reaches 193 in the first pack of rows. It is known that the row with the maximum non-zero elements bottlenecks the performance when a pack of rows is

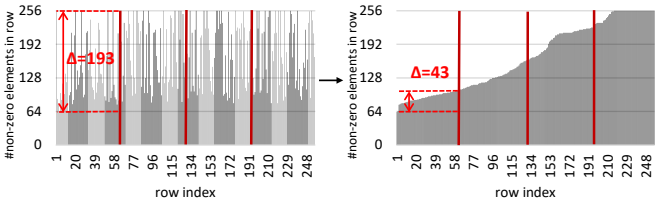

Fig. 5. The number of non-zero elements in each row of the attention mask.

allocated to the systolic array. It results in a large number of idle cells and incurs unacceptable waiting latency. To solve this problem, we rearrange the rows of the $hmask$, which packs the rows with similar sparsity for execution. The rearranged rows are visualized in Fig. 5(b), the maximum difference in the 1st pack decreases to 43, which means the rows exhibit similar progress and achieve the highest resource utilization.

To implement this method, we reuse the insertion-sort array in the transition-down unit to sort rows of the $hmask$ based on its sparsity, which is defined as the number of non-zero elements in each row. Then, the sort indices are stored in the buffer of the tensor unit, directing the gathering order of Q vectors. After completing the multiplications, the results are written back to memory based on the sort indices. As a consequence, the workload becomes balanced and the performance gains are maximized.

V. Evaluation

A. Evaluation Setup

1) Benchmarks: We evaluate Sava on 4 popular point cloud transformers across extensive datasets as presented in Table. I.

TABLE I
Benchmarks and Datasets

Models	Training Set	Test Set	Application
PCT [7]	ModelNet40	ModelNet40	Classification
PoinTr [8]	PCN [9]	KITTI	Shape Completion
PUT [10]	PU_GAN [11]	PU_GAN	Upsampling
Point-Bert [12]	ShapeNet55	ModelNet40	Classification

2) Architectural Modeling: We develop a cycle-accurate simulator to model the behavior of our Sava architecture. We integrate DRAMSim3 as the external memory to obtain accurate and reliable memory access latency. To provide sufficient memory bandwidth, we set up the DRAM as HBM2 with a capacity of 8GB. Its area and power are obtained from the Verilog workbench generated by DRAMSim3. The total capacity of on-chip buffers is 86.2KB, and their area and power are modeled through CACTI [13]. Within the tensor unit, we employ a 64×32 2-bit array along with a 64×15 32-bit array, where each row supports up to 75% sparsity of a 64×64 sparse block. Furthermore, the Sava is verified through RTL simulations and synthesized under 28nm technology on 1GHz frequency. Its total area is $6.657mm^2$.

3) Baselines: We compare our Sava with the baseline model implemented on CPU (Intel Xeon Gold 6226R), GPUs (NVIDIA A100 PCIe 80GB, NVIDIA Jetson AGX Xavier) and the state-of-the-art ASICs (PointAcc, PRADA), with each configured with equivalent computing resource for fairness.

B. Accuracy

To validate the accuracy of the spatial- and value-aware hybrid pruning strategy, we implement our algorithms using PyTorch on GPU across various benchmarks. The experimental results are depicted in Table. II. Our method creates significant sparsity in attention only with negligible accuracy loss. Specifically, we have pruned 59.03% values on average, indicating a substantial computation reduction in attention.

TABLE II
Accuracy of the Hybrid Pruning Strategy

Models	Metrics	Result	Loss	Sparsity
PCT	Accuracy	93.03%	0.24%	63.25%
PoinTr	MMD [14]	0.000306	0.55%	61.39%
PUT	CD-*l2* [15]	0.001407	0.26%	64.99%
Point-Bert	Accuracy	92.75%	0.12%	48.12%

C. Speedup and Energy Savings

Fig. 6 and 7 present the speedup and energy savings of Sava over CPU, GPUs and ASICs. On average, our Sava achieves 10.3×, 3.6×, 3.3×, 2.6×, 2.2× speedup and 20×, 8.8×, 6.9×, 3.2×, 2.4× energy savings, respectively. Notably, Sava provides the highest performance on PoinTr. This is attribute to our Sava primarily aiming to accelerate the back-end transformers, and PoinTr, with the most attention layers in both the encoder and decoder networks, can take full benefit from our approach. Point-Bert is another attention-dominant model, and it achieves satisfactory accuracy when employing lower precision (2-bit) quantization in its 2nd and 3rd layers. Thus, Our Sava offers remarkable performance on Point-Bert similar to that of PoinTr. While the transition-down stage bottlenecks PUT (see Fig. 1), which shows fewer performance gains from Sava in comparison with the other models. Fortunately, our Sava is orthogonal to many state-of-the-art PCNN accelerators like PointAcc, which has achieved good performance in front-end acceleration. We will further discuss this point in the following subsection.

D. Exploration

*1) **An ablation study of the data rearrangement mechanism:*** To address the workload imbalance in unstructured sparse computations, we introduce a data rearrangement mechanism to pack rows with similar sparsity, improving resource utilization and reducing overall latency. To validate its effectiveness, we designed an ablation study, as shown in Fig. 8. It demonstrates that the data rearrangement mechanism offers 2.9× speedup and 57.45% improvement in resource utilization on average. The results indicate the significant potential of the data rearrangement mechanism in optimizing resource utilization and reducing latency for unstructured sparse computations. Moreover, our mechanism serves as a valuable reference for solving the workload imbalance issues in general unstructured sparse operations.

*2) **The generality of Sava:*** As we mentioned in Section I, the front-end of the point cloud transformer is essentially a layer of the traditional PCNN. Numerous previous works focus on accelerating the front-end and achieving notable performance,

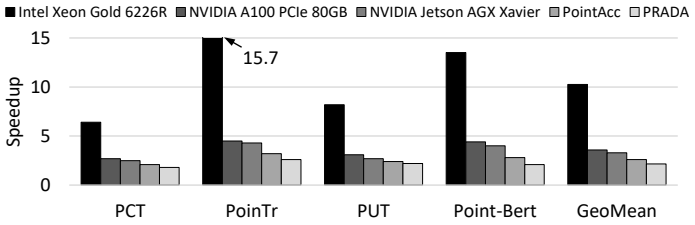

Fig. 6. The speedup of Sava over CPU, GPUs and ASICs.

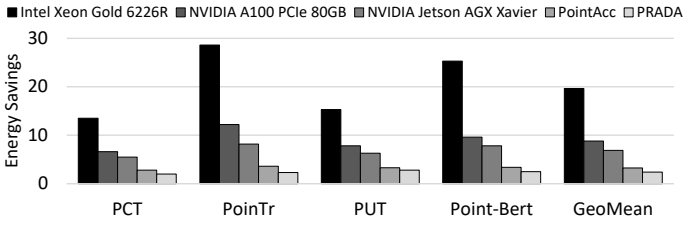

Fig. 7. The energy efficiency of Sava over CPU, GPUs and ASICs.

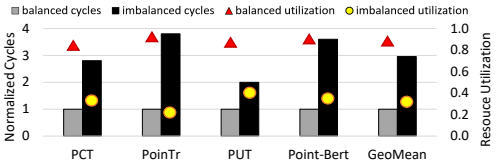

Fig. 8. An ablation study of the data rearrangement mechanism.

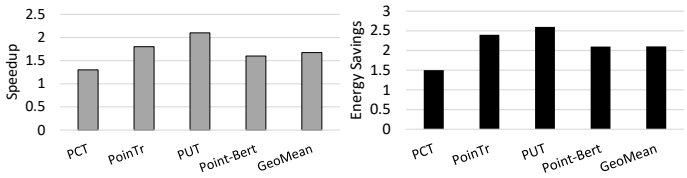

Fig. 9. The speedup and energy efficiency of Sava+PointAcc over Sava.

which is perfectly orthogonal to our Sava. We carry out an experiment to combine Sava with pointAcc. To be specific, we reserve PointAcc's Mapping Unit (MPU) for front-end processing, replace its Fetch-on-Demand computation flow with the Gather-MatMul-Scatter flow to avoid resource under-utilization, and then employ our configurable mixed-precision systolic array for the back-end. Fig. 9 presents the performance and energy efficiency of the combined architecture in comparison with Sava. It shows that PointAcc+Sava achieves 1.7× speedup and 2.1× energy savings on average. It further demonstrates that our Sava exhibits excellent generality, allowing it to be seamlessly integrated with previous state-of-the-art methods to achieve significantly improved performance.

*3) **Variable precision quantization**:* As mentioned in Section III-B, we employ variable precision quantization across different layers. Essentially, adopting lower precisions in quantization may result in the toggling of some mask bits, which is equivalent to introducing noise into the attention. If the attention layer exhibits stronger noise resistance, it should be less sensitive to low-bit quantization. To measure the noise resistance, we introduce random noise to each layer separately and observe its accuracy loss. The more pronounced the changes in accuracy, indicating a higher sensitivity of the attention layer to noise, which necessitates the use of higher bit precision to ensure accuracy. As shown in Table. III, we determine the further pruned quantization precisions based on the sensitivity of layers, which results in an acceptable accuracy loss while achieving a significant reduction in computation.

VI. Conclusion

This paper first optimizes point cloud transformers through a customized algorithm-architecture co-design scheme. We discover the varying importance among different spatial regions and values in the point cloud transformer, which directs us to prune those redundant computations and memory accesses. Therefore, we propose a spatial- and value-aware hybrid pruning strategy. In support of the algorithms, we introduce Sava architecture, which employs an extensive transition-down module along with a mixed-precision systolic array to accommodate data with different bitwidths. We evaluate our Sava on several widely-used models, showing that the Sava achieves remarkable performance and accuracy than the SOTA PCNN accelerators.

TABLE III
Noise Sensitivity and Variable Precision Quantization

Models	Sensitive Layers	Quant Precisions/bit	Speedup
PCT	2	2,8,2,2	1.4×
PoinTr	2, 5	2,2,8,2,2,4	1.6×
PUT	5	2,2,2,2,8	1.1×
Point-Bert	1, 4	4,2,2,4	1.3×

VII. Acknowledgements

This work is partly supported by the National Natural Science Foundation of China (Grant No. 62202288, 61972242).

References

[1] F. et al, "Mesorasi: Architecture support for point cloud analytics via delayed-aggregation," in *MICRO*. IEEE, 2020, pp. 1037–1050.
[2] Z. et al, "Point transformer," in *CVPR*, 2021, pp. 16 259–16 268.
[3] Q. et al, "Geometric back-projection network for point cloud classification," *IEEE Transactions on Multimedia*, vol. 24, pp. 1943–1955, 2021.
[4] S. et al, "Prada: Point cloud recognition acceleration via dynamic approximation," in *DATE*. IEEE, 2023, pp. 1–6.
[5] Q. et al, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *NeurIPS*, vol. 30, 2017.
[6] L. et al, "Pointacc: Efficient point cloud accelerator," in *MICRO-54*, 2021, pp. 449–461.
[7] G. et al, "Pct: Point cloud transformer," *CVM*, vol. 7, pp. 187–199, 2021.
[8] Y. et al, "Pointr: Diverse point cloud completion with geometry-aware transformers," in *ICCV*, 2021, pp. 12 498–12 507.
[9] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert, "Pcn: Point completion network," in *2018 international conference on 3D vision (3DV)*. IEEE, 2018, pp. 728–737.
[10] S. Qiu, S. Anwar, and N. Barnes, "Pu-transformer: Point cloud upsampling transformer," in *Proceedings of the Asian Conference on Computer Vision*, 2022, pp. 2475–2493.
[11] L. et al, "Pu-gan: a point cloud upsampling adversarial network," in *ICCV*, 2019, pp. 7203–7212.
[12] Y. et al, "Point-bert: Pre-training 3d point cloud transformers with masked point modeling," in *CVPR*, 2022, pp. 19 313–19 322.
[13] M. et al, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.
[14] G. et al, "A kernel method for the two-sample-problem," *NeurIPS*, vol. 19, 2006.
[15] F. et al, "A point set generation network for 3d object reconstruction from a single image," in *CVPR*, 2017, pp. 605–613.