2021 IEEE Workshop on Signal Processing Systems (SiPS) | 978-1-6654-0144-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/SiPS52927.2021.00054

Exploration of Energy-Efficient Architecture for Graph-Based Point-Cloud Deep Learning

Jie-Fang Zhang and Zhengya Zhang
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, Michigan, USA
{jfzhang, zhengya}@umich.edu

***Abstract*—Deep learning on point clouds has attracted increasing attention in the fields of 3D computer vision and robotics. In particular, graph-based point-cloud deep neural networks (DNNs) have demonstrated promising performance in 3D object classification and scene segmentation tasks. However, the scattered and irregular graph-structured data in a graph-based point-cloud DNN cannot be computed efficiently by existing SIMD architectures and accelerators. Following a review of the challenges of point-cloud DNN and the key edge convolution operation, we provide several directions in optimizing the processing architecture, including computation model, data reuse, and data locality, for achieving an effective acceleration and an improved energy efficiency.**

***Index Terms*—Point cloud, neural network, edge convolution, graph convolution, spatial locality, graph traversal**

I. Introduction

3D deep learning has received growing interest in recent years due to its widespread applications in the 3D space, including indoor navigation, object classification, scene segmentation, and shape synthesis [1]–[4]. A 3D space can be accurately represented by a point cloud. A point cloud can also be acquired directly from most 3D data acquisition devices like LiDARs and IR sensors. It is natural to consider applying deep learning to point cloud data.

Following the success of deep neural network (DNN) and convolutional neural network (CNN) on 2D image applications [5]–[7], researchers have worked on applying the concept of CNNs to the point clouds in the 3D space. Past works [8]–[11] tried to project point clouds or convert them to voxels in a 3D space. More recently, attempts were made [12]–[14] to process point clouds directly to achieve a higher performance. Among these, graph-based methods [13], [15], [16] were proposed to extract local neighborhood information between the points and integrate them into the global shape structure to improve performance. A prime example is DGCNN [13] that uses a graph-based operator called edge convolution (EdgeCONV) to integrate both local and global features.

Fig. 1 shows the DGCNN architecture and illustrates the EdgeCONV operation. An EdgeCONV is done in two steps. In the first step, a K-nearest neighbor (KNN) graph is constructed from the input point cloud to uncover the spatial relationship between points. In the second step, following the KNN graph edges, a graph convolution (GraphCONV) is applied to compute the local neighborhood features and the global feature of every vertex point. The local neighborhood features and

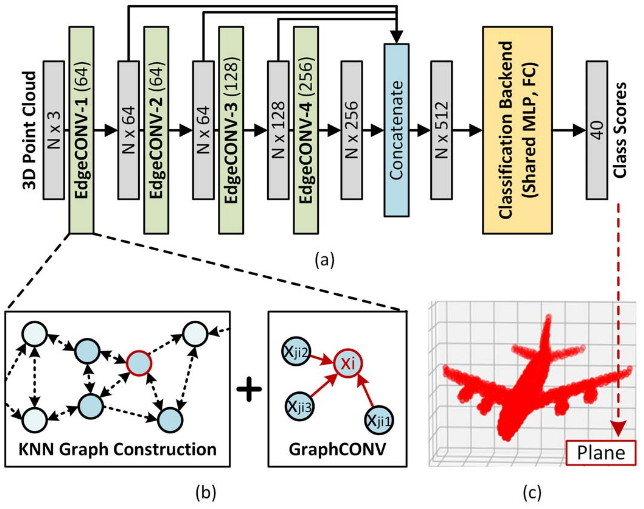

Fig. 1. Illustration of (a) DGCNN's network architecture [13], (b) EdgeCONV layer consisting of KNN graph construction and GraphCONV on vertex point i, and (c) 3D object classification example.

the global feature are summed to obtain the output feature. EdgeCONV has become a popular operator [17]–[19] even beyond DGCNN. Various forms of EdgeCONV have been employed in many recent point-cloud DNN works [20], [21].

In this paper, we present the challenges of implementing EdgeCONV, the key operator and building block of graph-based point-cloud DNNs. We outline some promising directions to optimize EdgeCONV computation to increase computational parallelism and reduce unnecessary data movement, thereby enabling a higher performance and efficiency.

II. Edge Convolution and Challenges

Suppose an input point cloud $\boldsymbol{X}$ contains N points $\boldsymbol{x}_0, ..., \boldsymbol{x}_{N-1}$, where $\boldsymbol{x}_i \in \mathbb{R}^C$ is the i-th point feature and C is the dimension of the input feature space. Given a point cloud, the EdgeCONV layer can be understood as projecting each point's feature from an input C-dimensional space to an output F-dimensional space. As mentioned above and illustrated in Fig. 1(b), an EdgeCONV operation is done in two steps: 1) KNN graph construction and 2) GraphCONV on the constructed KNN graph. The steps are described below.

A. KNN Graph Construction

A KNN graph from $\boldsymbol{X}$ is constructed as a directed graph $G_K = (V, E)$, where $V = \{0, \dots, N-1\}$ are the vertices representing the points in the point cloud, and $E = \{(i, j_{i1}), \dots, (i, j_{iK}), \dots\}$ for $0 \le i, j_{ik} < N$ and $1 \le k \le K$ are the edges. An edge $(i,\ j_{ik})$ connects the vertex point i to its k-th nearest neighbor. The Euclidean distance is used to construct the KNN graph [13] as described in Eq. (1). By definition, each vertex point is also neighbor to itself.

$$j_{ik} = \underset{j\in[0,N),k}{\arg\min}\ D(i,j) = \underset{j\in[0,N),k}{\arg\min}\ \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 . \tag{1}$$

B. Graph Convolution

GraphCONV is defined as the weighted sum between the global shape structure and the local neighborhood structures [13]. Given the model weights $\boldsymbol{\phi}_0, \dots, \boldsymbol{\phi}_{F-1}, \boldsymbol{\theta}_0, \dots, \boldsymbol{\theta}_{F-1} \in \mathbb{R}^C$, the f-th output feature of vertex point i after GraphCONV is defined by

$$\boldsymbol{x}'_{if} \quad = \quad \max_{1\le k\le K} \mathrm{ReLU}\left(\boldsymbol{\phi}_f \cdot \boldsymbol{x}_i + \boldsymbol{\theta}_f \cdot (\boldsymbol{x}_{j_{ik}} - \boldsymbol{x}_i)\right). \tag{2}$$

Simply put, the $\boldsymbol{\phi}$ weights are applied to the vertex point's feature $\boldsymbol{x}_i$ to compute the global feature partial sum (fpsum), and $\boldsymbol{\theta}$ weights are applied to the differences between the vertex point and its neighbors $(\boldsymbol{x}_{j_{ik}} - \boldsymbol{x}_i)$ to compute a local fpsum. The two parts are summed before a ReLU operation. The max selects the most salient output fpsum among the K options as the output feature.

C. Computational Challenges

In its mathematical form, GraphCONV bears some similarities with computations explored in the past, but the EdgeCONV computation, comprising both KNN graph construction and GraphCONV, cannot be supported efficiently by existing hardware solutions.

In EdgeCONV, the KNN graph is constructed in runtime and the edges are not known beforehand. Without knowing the vertices in advance, they cannot be organized ahead of time, so the neighbors of a vertex may be randomly dispersed in the system memory. Conventional general-purpose SIMD architectures, i.e., CPUs and GPUs, have no option but to fetch the randomly scattered graph vertices and gather them for vector computation, resulting in a low compute utilization and a low efficiency. Existing DNN/CNN accelerators [22], [23] are designed for parallel computation on regular-structured data, e.g., 2D images, which are well-organized and can be accessed sequentially from memory. Presented with randomly scattered data, a DNN/CNN accelerator will lose performance.

On the other hand, existing graph processing accelerators [24], [25] are designed for irregular data accesses. Without considering data reuse and parallel processing, it is unlikely for a graph processing accelerator to produce a high efficiency. Recently, accelerators for graph neural network or graph convolutional network (GNN/GCN) [26], [27] were proposed to handle both irregular-structured data and parallel computation for GCN. However, they lack runtime graph construction capability, and thus cannot fully support EdgeCONV. The GCN accelerators also do not exploit the characteristics of point clouds or KNN graphs, making it impossible to provide the best performance and efficiency.

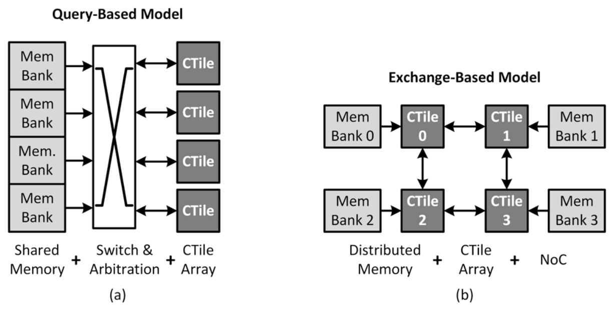

Fig. 2. EdgeCONV computation models: (a) query-based model, and (b) exchange-based model.

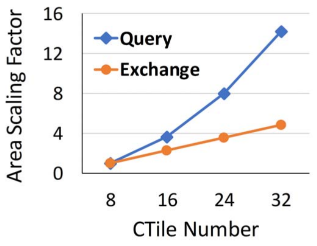

Fig. 3. Area scalability comparison between the query-based model and exchange-based model.

III. Directions for Architecture Optimization

After identifying the challenges of existing hardware architectures, we explore several directions to optimize the processing: namely distributing the computation, increasing the data reuse, and increasing the data locality, all aimed at boosting the processing performance and efficiency.

A. Computation Model

A conventional spatial accelerator design utilizes multiple compute tiles (CTiles) to distribute the computation and increase the parallelism. To parallelize EdgeCONV, i.e., KNN graph construction and GraphCONV, one option is to distribute the vertex points to the CTiles to allow them to compute independently. The distributed CTiles collectively construct the KNN graph and produce the GraphCONV results for a given input point cloud.

KNN graph construction can be easily distributed, but GraphCONV is a different story. The challenge with distributing GraphCONV is due to GraphCONV's access of neighbor points. The neighbors of a vertex point are not available until the KNN graph construction is done, and the neighbors can be randomly scattered in memory without a fixed pattern. The neighbor access challenge renders the distributed GraphCONV highly inefficient as parallel CTiles will be competing for memory bandwidth.

To analyze the performance implications and the best approach forward, we consider two distributed computation models: a query-based model and an exchange-based model as shown in Fig. 2. In an query-based model, all points are centrally placed in a shared memory between CTiles. As shown in Fig. 2(a), each CTile operates independently and makes requests to the shared memory controller to access neighbor points. The memory controller serves potentially multiple requests from CTiles at the same time. When a conflict arises, the memory controller arbitrates the requests from CTiles by granting one CTile access while stalling the other ones. The query-based model is simple to implement but its scalability is poor.

In an exchange-based model, points are distributed and stored locally in CTiles. Each CTile owns its memory bank, and it will try to access points locally to the maximum extent possible, as shown in Fig. 2(b). However, it is not possible to store all the points in each CTile's local memory, as the memory size would be too large. In case a CTile needs to access a neighbor point not available locally, a CTile requests the point from another CTile that holds the neighbor point. This case results in CTile-CTile communications. Such communications are typically faciliated by a network-on-chip (NoC) that links all CTiles. The exchange-based model circumvents the shared memory bottleneck and is potentially more scalable, but its bottleneck is transferred to the NoC. If a CTile needs a large number of neighbor points located outside of its local memory, the NoC will incur a large burden, presenting a performance bottleneck and a long latency.

Since data fetching is the critical part in both computation models, we compare the hardware (area) cost of the main components responsible for fetching neighbor points into each CTile: an all-to-all switch with arbiter for the query-based model, and a 2D mesh NoC for the exchange-based model. The hardware cost for different number of CTiles are obtained from synthesis and normalized to the 8-CTile baseline to obtain the area scaling factor. As shown in Fig. 3, the cost for a query-based design increases super-linearly with more CTiles, and large-size designs suffer from long critical paths and high overheads. An exchange-based design demonstrates a better scalability with almost linear area increase to the number of CTiles.

B. Data Reuse

In both the query-based model and the exchange-based model, a CTile is responsible for computing a set of vertex points. The computation of a vertex point requires the vertex point itself and its K neighbor points as described by Eq. (2). Due to the possible overlap of vertex points, different CTiles may be accessing overlapping neighbors and performing duplicate computation.

An alternative form of Eq. (2) can be obtained from Eq. (2) by reordering the steps of computation:

$$\boldsymbol{x}'_{if} = \mathrm{ReLU}(\max_{1\le k\le K}(\boldsymbol{\theta}_f \cdot \boldsymbol{x}_{j_{i_k}}) + (\boldsymbol{\phi}_f - \boldsymbol{\theta}_f)\cdot \boldsymbol{x}_i). \quad (3)$$

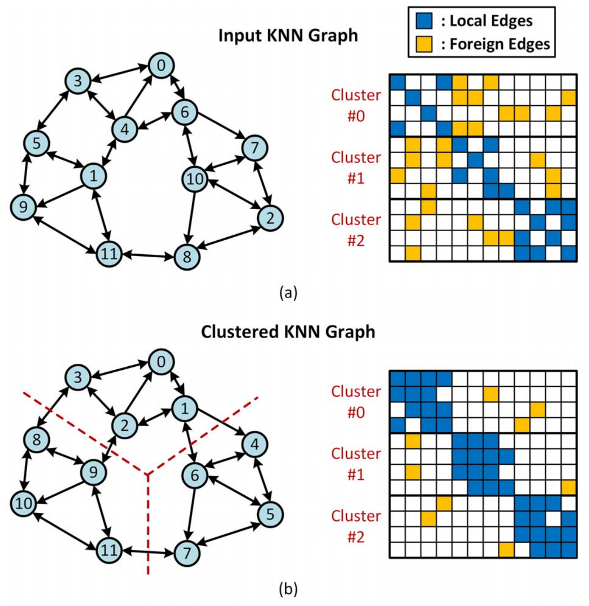

Fig. 4. (a) The KNN graph for an input point cloud and its adjacency matrix representation; (b) the clustered KNN graph after identifying subgraphs using clustering and its adjacency matrix representation; the vertices are renumbered to visualize the effects of clustering.

Based on this alternative form, each fpsum, $(\boldsymbol{\phi}_f - \boldsymbol{\theta}_f)\cdot \boldsymbol{x}_i$ or $\boldsymbol{\theta}_f \cdot \boldsymbol{x}_{j_{i_k}}$, can be computed by one CTile once, and if another CTile needs the fpsum, it can be transferred through the NoC to prevent duplicate dot product computation. Using the alternative form of Eq. (3), the number of dot products is cut by $\mathcal{O}(K)$ compared to the original form of Eq. (2).

C. Data Locality

Community structure exists in KNN graphs of real-world point clouds, where subsets of points tend to be more densely connected. By assigning such densely-connected clusters to a CTile, we can maximize the local access by a CTile. More specifically, we highlight two advantages of leveraging spatial locality in GraphCONV processing. First, the computational parallelism and latency of a CTile can be improved if more neighbor points can be accessed locally in the CTile at once. Second, the fetch efficiency of a CTile can be improved if the NoC transfers are minimized or shortened. However, the points stored in memory can be in any order, possibly following the order that the points are acquired. It is best to reorder the points before they are assigned to CTiles to maximize the locality.

Fig. 4(a) shows an example of a point cloud's KNN graph and its adjacency matrix representation. In the KNN graph, a directed edge from node j to node i indicates that the vertex point i is connected to its neighbor point j, and is represented by the entry (i, j) in the adjacency matrix. In Fig. 4(a), the points of consecutive point indices are grouped into three clusters as indicated. Each cluster is assigned to a CTile. From a CTile's perspective, if a neighbor point also belongs to the cluster, local access is available; and if a neighbor point

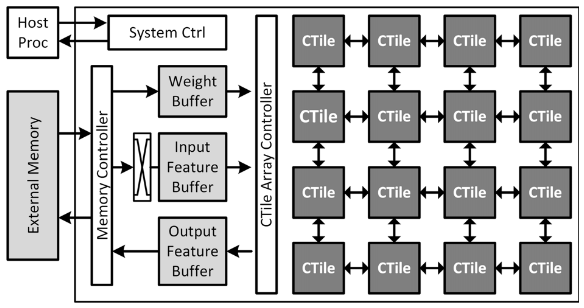

Fig. 5. EdgeCONV accelerator architecture.

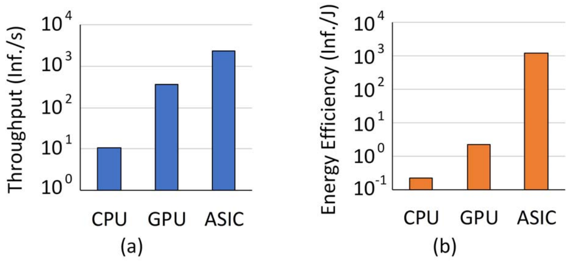

Fig. 6. Comparison between EdgeCONV accelerator (ASIC) and CPU and GPU: (a) performance comparison, and (b) energy efficiency comparison.

belongs to a different cluster, it needs to access it from a different CTile over the NoC. In the adjacency matrix, local connectivity is indicated by local edges and the non-local connectivity is indicated by foreign edges.

Graph traversal algorithms [28]–[30] can be used to reorder points to obtain better locality. In particular, a graph traversal algorithm can produce a new order of points. The KNN graph can be split into clusters following the new order. The result of a reordering is presented and visualized in Fig. 4(b). The reordering process itself can be viewed as shuffling the rows and columns of the adjacency matrix. The reordered adjacency matrix features more local edges and fewer foreign edges in each cluster, demonstrating the effectiveness of the approach.

Several common graph traversal algorithms, including breadth-first search (BFS), depth-first search (DFS), and bounded DFS (BDFS), were evaluated for their effectiveness in uncovering densely-connected subgraphs in the KNN graph [28]–[31]. The results indicate that the common traversal methods improve the probability that a neighbor point can be found in a local tile (hit probability) by 5.3× on average. Notably, BFS not only improves the hit probability, but also reduces the average transfer distance of non-local neighbor points, allowing the foreign edges in the adjacency matrix to be brought closer to the main diagonal compared to DFS or BDFS.

IV. EdgeCONV Accelerator and Evaluation

Combining the above approaches, an EdgeCONV accelerator architecture is constructed using BFS graph traversal to identify the dense subgraphs in the KNN graph. The accelerator architecture, shown in Fig. 5, is designed in a 4×4 CTile array with a total of 2,048 MACs for KNN graph construction and GraphCONV computation. Each CTile is connected to its neighbor CTiles to exchange neighbor fpsums through a mesh NoC. Once loaded on-chip, the input point features are distributed to pre-assigned CTiles following the cluster mapping.

The EdgeCONV accelerator is evaluated based on the EdgeCONV layers specified in [13] in the ModelNet40 [32] dataset, a widely-used point cloud dataset for 3D object classification tasks. The EdgeCONV accelerator is compared to commercial general-purpose computing platforms, i.e., CPU (Intel i7-7700K) and GPU (Nvidia GTX-1080Ti), to demonstrate its performance and energy efficiency. The CPU and GPU run at higher clock frequencies of 4.2 GHz and 1.48 GHz, respectively, and have an on-chip memory of 8 MB and 4 MB, respectively, to maximize on-chip buffering and data reuse. The CPU has 8 threads and the GPU has 3,584 cores for floating point computation.

Fig. 6 compares the EdgeCONV performance and energy efficiency of the EdgeCONV accelerator (ASIC) to the CPU and GPU baselines. For the CPU and GPU comparison targets, DGCNN was implemented in PyTorch following [13], and the latency and power consumption for the EdgeCONV layers were measured. The performance and power of the EdgeCONV accelerator were obtained from synthesis in a 28nm CMOS technology. In terms of throughput, measured in number of inferences per second (Inf./s), the EdgeCONV accelerator demonstrates an improvement of 204.7× and 6.2× compared to the CPU and GPU baselines, respectively. As for energy efficiency, measured in number of inferences per joule (Inf./J), the design shows an improvement of 5540.5× and 543.7× over the CPU and GPU baselines, respectively.

Beyond the general architectural approaches outlined here, we recently created innovations in parallel graph traversal, lightweight NoC, and streamlined dataflow [33] to further enhance the performance and efficiency of EdgeCONV processing on point clouds.

V. Conclusion

We provide an overview of the EdgeCONV computation, a key operator for supporting graph-based point-cloud DNNs. Three optimization approaches can be considered in designing an accelerator for EdgeCONV computation. First, the exchange-based model allows a more scalable design with a better fetch efficiency over the query-based model. Second, an alternative form of GraphCONV computation provides higher data reuse and removes redundant computation. Third, graph traversal can be applied to constructed KNN graphs to reorder points for better clustering to obtain locality. When evaluated on EdgeCONV layers in DGCNN, a prototype EdgeCONV accelerator provides significant speedup and energy efficiency improvement over a commercial GPU.

References

[1] Y. Guo *et al.*, "Deep learning for 3D point clouds: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2020.

[2] J. Zhang *et al.*, "A review of deep learning-based semantic segmentation for point cloud," *IEEE Access*, vol. 7, pp. 179 118–179 133, 2019.

[3] E. Ahmed *et al.*, "Deep learning advances on different 3D data representations: A survey," *arXiv preprint arXiv:1808.01462*, 2018.

[4] Y. Xie *et al.*, "Linking points with labels in 3D: A review of point cloud semantic segmentation," *IEEE Geosci. Remote Sens. Mag.*, 2020.

[5] A. Krizhevsky *et al.*, "ImageNet classification with deep convolutional neural networks," in *Proc. Conf. Neural Information Processing Systems (NeurIPS)*, 2012, pp. 1097–1105.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2015.

[7] K. He *et al.*, "Deep residual learning for image recognition," in *Proc. Conf. Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[8] H. Su *et al.*, "Multi-view convolutional neural networks for 3D shape recognition," in *Proc. Int. Conf. Computer Vision (ICCV)*, 2015, pp. 945–953.

[9] C. R. Qi *et al.*, "Volumetric and multi-view cnns for object classification on 3D data," in *Proc. Conf. Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 5648–5656.

[10] D. Maturana and S. Scherer, "VoxNet: A 3D convolutional neural network for real-time object recognition," in *Int. Conf. Intelligent Robots and Systems (IROS)*, 2015, pp. 922–928.

[11] P.-S. Wang *et al.*, "O-CNN: Octree-based convolutional neural networks for 3D shape analysis," *ACM Trans. on Graphics (TOG)*, vol. 36, no. 4, pp. 1–11, 2017.

[12] C. R. Qi *et al.*, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. Conf. Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 652–660.

[13] Y. Wang *et al.*, "Dynamic graph CNN for learning on point clouds," *ACM Trans. on Graphics (TOG)*, vol. 38, no. 5, pp. 1–12, 2019.

[14] C. R. Qi *et al.*, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in *Proc. Conf. Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5099–5108.

[15] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. Conf. Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 3693–3702.

[16] Y. Shen *et al.*, "Mining point cloud local structures by kernel correlation and graph pooling," in *Proc. Conf. Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4548–4557.

[17] K. Zhang *et al.*, "Linked dynamic graph CNN: Learning on point cloud via linking hierarchical features," *arXiv preprint arXiv:1904.10014*, 2019.

[18] H. You *et al.*, "PVNet: A joint convolutional network of point cloud and multi-view for 3D shape recognition," in *Proc. Int. Conf. Multimedia*, 2018, pp. 1310–1318.

[19] Y. Wang and J. M. Solomon, "Deep closest point: Learning representations for point cloud registration," in *Proc. Int. Conf. Computer Vision (ICCV)*, 2019, pp. 3523–3532.

[20] K. Hassani and M. Haley, "Unsupervised multi-task feature learning on point clouds," in *Proc. Int. Conf. Computer Vision (ICCV)*, 2019, pp. 8160–8171.

[21] C. Chen *et al.*, "GAPNet: Graph attention based point neural network for exploiting local feature of point cloud," *arXiv preprint arXiv:1905.08705*, 2019.

[22] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. Int. Symp. Computer Architecture (ISCA)*, 2017, pp. 1–12.

[23] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits (JSSC)*, vol. 52, no. 1, pp. 127–138, 2016.

[24] M. M. Ozdal *et al.*, "Energy efficient architecture for graph analytics accelerators," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 166–177, 2016.

[25] T. J. Ham *et al.*, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," in *Proc. Int. Symp. Microarchitecture (MICRO)*, 2016, pp. 1–13.

[26] M. Yan *et al.*, "HyGCN: A GCN accelerator with hybrid architecture," in *Proc. Int. Symp. High-Performance Computer Architecture (HPCA)*, 2020, pp. 15–29.

[27] K. Kiningham *et al.*, "GRIP: a graph neural network accelerator architecture," *arXiv preprint arXiv:2007.13828*, 2020.

[28] J. Banerjee *et al.*, "Clustering a dag for cad databases," *IEEE Trans. Softw. Eng.*, vol. 14, no. 11, pp. 1684–1699, 1988.

[29] P. Yuan *et al.*, "PathGraph: A path centric graph processing system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 10, pp. 2998–3012, 2016.

[30] A. Mukkara *et al.*, "Exploiting locality in graph analytics through hardware-accelerated traversal scheduling," in *Proc. Int. Symp. Microarchitecture (MICRO)*, 2018, pp. 1–14.

[31] C. E. Leiserson and T. B. Schardl, "A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers)," in *Proc. Symp. Parallelism in Algorithms and Architectures*, 2010, pp. 303–314.

[32] Z. Wu *et al.*, "3D ShapeNets: A deep representation for volumetric shapes," in *Proc. Conf. Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1912–1920.

[33] J.-F. Zhang and Z. Zhang, "Point-X: A spatial-locality-aware architecture for energy-efficient graph-based point-cloud deep learning," *54th IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, In Press.