# Point-X: A Spatial-Locality-Aware Architecture for Energy-Efficient Graph-Based Point-Cloud Deep Learning

Jie-Fang Zhang
jfzhang@umich.edu
University of Michigan, Ann Arbor
Ann Arbor, Michigan, USA

Zhengya Zhang
zhengya@umich.edu
University of Michigan, Ann Arbor
Ann Arbor, Michigan, USA

## 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. We present Point-X, an energy-efficient accelerator architecture that extracts and exploits the spatial locality in point cloud data for efficient processing. Point-X uses a clustering method to extract fine-grained and coarse-grained spatial locality from the input point cloud. The clustering maps the point cloud into distributed compute tiles to maximize intra-tile computational parallelism and minimize inter-tile data movement. Point-X employs a chain network-on-chip (NoC) to further reduce the NoC traffic and achieve up to 3.2× speedup over a traditional mesh NoC. Point-X's multi-mode dataflow can support all common operations in a graph-based point-cloud DNN, i.e., edge convolution, shared multi-layer perceptron, and fully-connected layers. Point-X is synthesized in a 28nm technology and it demonstrates a throughput of 1307.1 inference/s and an energy efficiency of 604.5 inference/J on the DGCNN workload. Compared to the Nvidia GTX-1080Ti GPU, Point-X shows 4.5× and 342.9× improvement in throughput and efficiency, respectively.

## CCS CONCEPTS

• **Computer systems organization → Neural networks**; **Data flow architectures**; • **Hardware → Hardware accelerators**.

## KEYWORDS

Point cloud, neural network, edge convolution, graph convolution, spatial locality, graph traversal

**ACM Reference Format:**
Jie-Fang Zhang and Zhengya Zhang. 2021. Point-X: A Spatial-Locality-Aware Architecture for Energy-Efficient Graph-Based Point-Cloud Deep Learning. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'21), October 18–22, 2021, Virtual Event, Greece.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3466752.3480081


*MICRO'21, October 18–22, 2021, Virtual Event, Greece*

ACM ISBN 978-1-4503-8557-2/21/10…$15.00
https://doi.org/10.1145/3466752.3480081

## 1 INTRODUCTION

3D deep learning has attracted increasing attention in recent years due to its wide applications in the 3D space, including indoor navigation, object classification, scene segmentation, shape synthesis and modeling. Among all 3D representations, point cloud has gained popularity since it shows an accurate representation of the real world and can be acquired directly as the raw output from most 3D data acquisition devices like LiDARs and IR sensors [12, 59, 1, 52]. The raw point clouds undergo common preprocessing steps, including background filtering, noise removal, and region of interest (ROI) identification, and the preprocessed ROI frames are then ready for the point-cloud processing.

After the wide success of deep neural network (DNN) and convolutional neural network (CNN) on 2D image applications [19, 40, 15, 43], researchers have worked on converting the insights from CNNs to the point clouds in the 3D space. The first attempts processed point clouds indirectly using an intermediate representation, i.e., multi-view [42, 57, 34] or volumetric [27, 36, 45]. These methods either project point clouds into several 2D images of different angles or convert them into voxels in a 3D grid, before applying well-established 2D or 3D CNNs to accomplish indirect point-cloud processing. However, these approaches were unable to capture the fine details and textures due to the data truncation during representation conversion. PointNet [33] proposed a point-based network to process point clouds directly without any projection or voxelization. However, PointNet only considers the global shape structure and not the relations between the points, thus it is unable to capture finer details, limiting its performance [35, 48].

To overcome the limitations, recent works propose to extract local neighborhood information by graph-based methods [39, 37, 48], and integrate them into the global shape structure to improve the performance over the PointNet model. In particular, DGCNN [48] uses a graph-based operator called edge convolution (EdgeCONV) to integrate both local and global features. Figure 1 shows the point cloud recognition pipeline using the DGCNN architecture and illustrates the principles of EdgeCONV. An EdgeCONV first uncovers the spatial relationship of points by constructing a K-nearest neighbor (KNN) graph of the input point cloud. Following the KNN graph edges, a graph convolution (GraphCONV) aggregates the local neighborhood features and the global feature of every vertex point to produce the output feature. The promising DGCNN results make EdgeCONV a widely adopted operator [60, 6, 56, 47].

Variants of EdgeCONV have emerged and are employed widely in recent graph-based point-cloud DNNs [14, 22, 5]. These variants share EdgeCONV's computation pattern which consists of 1) graph construction to uncover the spatial relationship between independent data points, followed by 2) convolution on the constructed

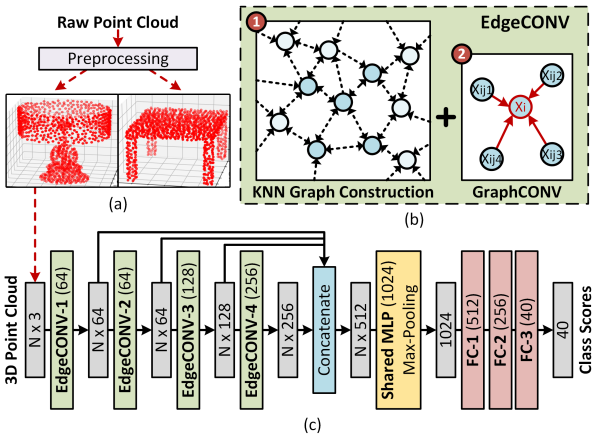


**Figure 1: Illustration of (a) point cloud recognition pipeline, (b) EdgeCONV layer divided into KNN graph construction and GraphCONV on vertex point *i*, and (c) DGCNN architecture for 3D object classification [48].**

graph to extract features for recognition. As such, the exploration and findings on EdgeCONV are applicable to these variants.

EdgeCONV computation cannot be supported efficiently by existing computing solutions. In computing EdgeCONV, a vertex's neighbors may be randomly dispersed in the system memory. Conventional SIMD architectures, i.e., CPUs and GPUs, have to fetch scattered graph vertices for vector computation, resulting in low compute utilization and efficiency. DNN/CNN accelerators [16, 8, 28, 38] are incapable of performing such computation because they are purposely designed for sequential memory access and regular-structured data, i.e., 2D image or 1D sequence. Graph processing accelerators [13, 31] are optimized for irregular data accesses and cannot fully exploit the data reuse or provide the computational parallelism needed in graph-based networks. Graph convolutional neural network (GNN/GCN) accelerators [54, 18, 11, 7, 10, 41, 24], handle both regular and irregular computation in GCNs. Different from the common GNNs where the graph structures are static and known in advance, EdgeCONV operates on dynamic graphs that are constructed in runtime. Most previous GNN works [54, 18, 11, 7, 10] focused on static graphs, and did not fully support EdgeCONV due to the lack of runtime graph construction. Recently, [41, 24] extended the support to dynamic graphs. However, they did not consider the community structures of point clouds, and are unable to provide the best efficiency.

To design a practical architecture for graph-based point-cloud processing, three objectives need to be addressed: *fetch efficiency*: to deliver a maximal number of neighbor points to compute under a limited transfer bandwidth; *computation efficiency*: to provide high compute parallelism and utilization for irregular point cloud workloads; and *flexibility*: to support diverse computation types in graph-based point-cloud networks.

We present Point-X, a spatial-locality-aware accelerator architecture for efficient graph-based point-cloud processing. Point-X extracts and leverages the spatial locality in the input point cloud to increase computational parallelism and reduce communication overhead, enabling higher fetch and computation efficiency. The main contributions are:

- A speculative breadth-first search (SBFS) graph traversal method is proposed to extract the spatial locality in the input point cloud. It achieves up to 9.2× faster execution over a conventional BFS graph traversal. A spatial-locality-aware clustering based on SBFS traversal is used to distribute input points into compute tiles for efficient processing.
- A lightweight chain NoC architecture is designed to leverage the spatial locality to effectively reduce the inter-tile traffic and its latency. The chain NoC design demonstrates a 3.2× shorter latency than a mesh NoC while incurring much lower area and energy overheads.
- A flexible compute tile and a multi-mode dataflow are designed to support all the common operations in graph-based point-cloud networks, including EdgeCONV, shared multi-layer perceptrons (MLPs), and fully-connected (FC) layers.

A Point-X design is synthesized in a 28nm technology. The design is estimated to occupy an area of 6.8 $mm^2$ and operate at a 1.0 GHz clock frequency. Point-X demonstrates an average speedup of 7.7× for GraphCONV over a baseline accelerator, and up to 12.1× higher energy efficiency in EdgeCONV over existing accelerators. Point-X achieves an end-to-end throughput of 1307.1 inference/s (Inf./s) and an energy efficiency of 604.5 inference/J (Inf./J) in running DGCNN for 3D object classification [48]. Compared to a general-purpose GPU and CPU, Point-X demonstrates a throughput improvement of 4.5×, 129.7×, respectively, and an energy efficiency improvement of 342.9× and 3160.9×, respectively.

## 2 BACKGROUND

An EdgeCONV layer in a point-cloud DNN projects the $N$ points from an input $C$-dimensional space into an output $F$-dimensional space. The input point cloud can be described as $\boldsymbol{X} = \{\boldsymbol{x}_0, ..., \boldsymbol{x}_{N-1}\}$, where $\boldsymbol{x}_i \in \mathbb{R}^C$ is the feature of $i$-th point, and $C$ is the dimensionality of the feature space. For instance, the first EdgeCONV layer receives an input of $C = 3$ which represent the $(x, y, z)$ coordinates in a 3D space. As illustrated in Figure 1(b), an EdgeCONV operation is divided into two steps: 1) KNN graph construction and 2) GraphCONV on the KNN graph.

### 2.1 Edge Convolution Computation

***KNN Graph Construction***. Given an input point cloud $\boldsymbol{X}$, a directed graph is constructed: $G_{(\boldsymbol{X},K)} = (V, E)$ with self-loops, where $V = \{0, \ldots, N-1\}$, $E = \{(i, j_{i1}), \ldots, (i, j_{iK}), \ldots\}$ for $i, j_{ik} \in V$, and $j_{ik}$ represents the $k$-th nearest neighbor to the vertex point $i$. The $k$-th nearest point to vertex point $i$ is found based on the Euclidean distance as described in Eq. (1).

$$j_{ik} = \underset{j \in V}{\arg\min}_k \, D(i, j) = \underset{j \in V}{\arg\min}_k \, \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|^2 . \tag{1}$$

***Graph Convolution***. In GraphCONV, the vertex point $i$ is convolved with its $K$ neighbors $j_{i1}, \ldots, j_{iK}$. In DGCNN [48], GraphCONV is defined as the combination of the global shape structure captured by the vertex point's feature $\boldsymbol{x}_i$ and local neighborhood structures captured by the differences between the vertex point and its neighbors $(\boldsymbol{x}_{j_{ik}} - \boldsymbol{x}_i)$. With learnable weights

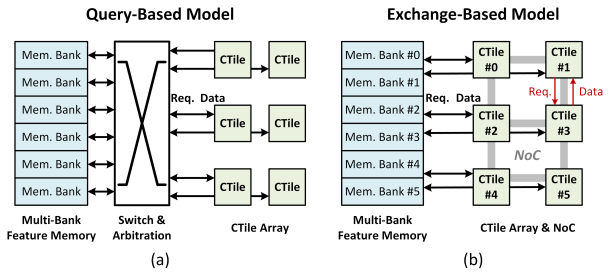


**Figure 2: EdgeCONV computation models: (a) query-based model, and (b) exchange-based model.**

$\boldsymbol{\phi}_0, \ldots, \boldsymbol{\phi}_{F-1}, \boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_{F-1} \in \mathbb{R}^C$, the $f$-th output feature of GraphCONV for the point $i$ is defined by

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

where $\boldsymbol{\phi}$ weights are applied to the vertex point to compute the global feature partial sum (fpsum), and $\boldsymbol{\theta}$ weights are applied to the difference to a neighbor point to compute a local fpsum. A ReLU operation is applied to the sum of the global fpsum and the local fpsum to generate an output fpsum. $K$ output fpsums are aggregated by a max operation to produce an output feature.

## 2.2 Computation Models and Bottlenecks

KNN graph construction is embarrassingly parallel and can be supported easily by SIMD and conventional spatial architectures. However, GraphCONV cannot be parallelized easily for two reasons: 1) there is no knowledge of a vertex point's neighbors prior to the KNN graph construction, limiting scheduling opportunity; and 2) the neighbors are randomly scattered in memory without a fixed pattern, limiting prefetch opportunity. These two factors result in highly inefficient computation when operating with the practical limitations of memory bandwidth, prefetch and scheduling capability. These limitations prohibit the parallel computation on multiple neighbors.

A spatial architecture utilizes multiple compute tiles (CTiles) for parallel computation. To parallelize KNN graph construction and GraphCONV, one possibility is to assign vertex points to independent CTiles. Under this setup, we present two EdgeCONV computation models, a query-based model and an exchange-based model, which differ in the fetch mechanisms and computation dataflow for GraphCONV computation. Figure 2 presents the two computation models for EdgeCONV operation. In both models, a multi-banked system memory is used to hold the point features and to support access by multiple CTiles.

***Query-Based Model***. The query-based model follows a direct parallelization of Eq. (2). In this model, each CTile operates independently and sees the memory as a single shared memory. A CTile is assigned a set of vertex points. To access neighbor points that are not available locally, a CTile sends requests to the centralized memory controller. The controller arbitrates the requests from all CTiles. Although the query-based model is straightforward, the all-to-all switch and arbiter design can be complex with poor scalability. Furthermore, frequent memory access conflicts occur when multiple CTiles request access to the same memory bank, resulting in a low fetch efficiency.

**Table 1: GraphCONV Computation Comparison ($F$ kernels, $N$ points, and $K$ neighbors per point)**

| Computation | Original Form, Eq. (2) | Reuse Form, Eq. (3) |
|---|---|---|
| Dot-Product $(\cdot)$ | $F \times N + F \times N \times K$ | $2 \times F \times N$ |
| Max-Pool (max) | $F \times N \times K$ | $F \times N \times K$ |
| Summation (+) | $F \times N \times K$ | $F \times N$ |
| ReLU | $F \times N \times K$ | $F \times N$ |

***Fpsum Reuse***. Different CTiles in the query-based model may be requesting and performing dot-product (DP) on overlapping neighbor points, causing duplicated computation during GraphCONV operation. We reformulate Eq. (2) to reduce computation redundancy and increase fpsum reuse. Starting from the original form in Eq. (2), we separate the DP of the vertex point from the neighbor points; and reorder the max and ReLU operations as shown below.

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

Following Eq. (3), the **vertex fpsums** (DP of the point with $(\boldsymbol{\phi} - \boldsymbol{\theta})$ weights) and the **neighbor fpsums** (DP of the point with $\boldsymbol{\theta}$ weights) can be computed once, cached and reused to prevent redundant DPs. Table 1 shows the comparison of computation of the two forms of GraphCONV defined by Eq. (2) and Eq. (3). Using the optimized form, the number of DP $(\cdot)$, summation (+), and ReLU operations are reduced by a factor of $(K + 1)/2$ or $K$.

***Exchange-Based Model***. Following Eq. (3), the exchange-based model allows CTiles to exchange neighbor fpsums through a NoC. In this model, each CTile is associated to a memory bank for accessing points locally to compute vertex fpsums and neighbor fpsums. To access a neighbor fpsum not available locally, a CTile requests from the CTile that holds the neighbor fpsum over the NoC. The exchange-based model cuts all redundant neighbor fpsum computation. However, a CTile may experience a longer transfer latency. Furthermore, the feature exchange may overwhelm the NoC bandwidth, resulting in an even lower fetch efficiency.

# 3 SPATIAL-LOCALITY-AWARE CLUSTERING

Prior works [29, 30, 4, 58, 49, 55, 7, 46] exploited the community structure of real-world graphs to improve locality for graph applications. Similarly, KNN graphs of real-world point clouds exhibit community structure where groups of points close in space form densely connected subgraphs.

Based on an exchange-based model, we review two types of beneficial spatial locality: fine-grained and coarse-grained. The fine-grained spatial locality refers to the case that the neighbor points are accessed from the *same* CTile (cluster) in computing GraphCONV of a vertex point. The coarse-grained spatial locality refers to the case that the neighbor points are accessed from *nearby* CTiles (clusters). The fine-grained spatial locality maximizes computational parallelism of a CTile by having most neighbor points in its local memory bank. The coarse-grained spatial locality helps reduce the data movement between CTiles by having the foreign

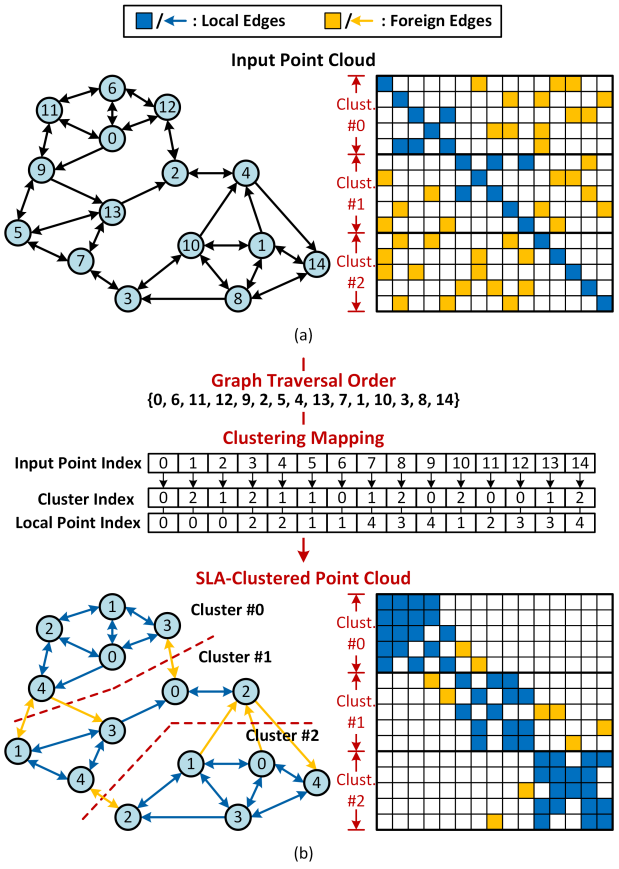


**Figure 3: (a) KNN graph of input point cloud and its adjacency matrix representation; (b) the KNN graph is traversed and the points clustered following traversal order; the clustered KNN graph and its adjacency matrix are shown.**

neighbor points needed by a CTile located in nearby CTiles. We present a clustering technique to extract both the fine-grained and coarse-grained spatial locality given a point cloud's KNN graph.

## 3.1 Graph Traversal for Spatial Locality

An example of a point cloud's KNN graph is shown in Figure 3(a). The nodes are numbered during point cloud acquisition and preprocessing, and the graph is represented by an adjacency matrix. 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. Note that, each point is also neighbor to itself. In Figure 3(a), the points of consecutive point indices are grouped into three clusters as illustrated on the adjacency matrix. An edge in the adjacency matrix is a local edge if both the vertex and the neighbor point belong to the same cluster or a foreign edge otherwise. In an SLA architecture, each cluster of points is assigned to a CTile. A local edge indicates local data access, and a foreign edge indicates access from a foreign cluster. The number of foreign edges suggests the amount of inter-tile data movement and the clustering efficiency.

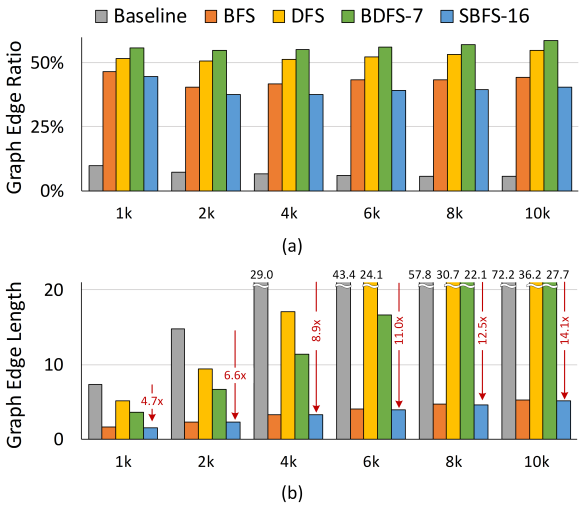


**Figure 4: Clustering performance of KNN graphs: (a) graph edge ratio, and (b) graph edge length, using different graph traversal methods on 1k to 10k point clouds.**

Our SLA clustering aims to maximize both fine-grained and coarse-grained spatial locality by taking advantage of the spatial relationship of points using graph traversals. The steps and result of SLA clustering are presented in Figure 3(b). Following the graph traversal order, the SLA clustering assigns every point into a cluster by mapping its point index to a pair of {cluster index, local point index}. Each cluster comprises a densely connected subgraph with improved fine-grained spatial locality, which can be visualized by the more local edges and fewer foreign edges in each cluster. As the scattered foreign edges are brought closer to the matrix diagonal, the coarse-grained spatial locality is also improved.

Graph traversal methods help uncover the spatial relationship of nodes in a graph [4, 58, 29]. To evaluate the capability of different graph traversal methods, we define two metrics: graph edge ratio to measure the fine-grained spatial locality, and graph edge length to measure the coarse-grained spatial locality. For every vertex point in a cluster, the **graph edge ratio** is defined as the proportion of local edges over all edges. The **graph edge length** measures the average distance (i.e., number of clusters away) between a vertex point and its foreign neighbor point. For instance, in Figure 3(a), the foreign edge from point 10 in cluster 2 to point 4 in cluster 0 has a distance of 2.

The graph edge ratio and the graph edge length are plotted in Figure 4 for common graph traversal methods, breadth-first search (BFS), depth-first search (DFS), bounded DFS (BDFS) [29] with a depth limit, evaluated using point clouds ranging from 1k to 10k points and compared to the input KNN graph baseline. As shown in Figure 4(a), for 1k points, all traversal methods result in a significant graph edge ratio improvement of 4.8 to 5.8× compared to the baseline. In particular, BDFS provides the highest graph edge ratio of 56%. With increased point sizes, the baseline shows a decrease in graph edge ratio, i.e., only 5.5% at 10k points, while the traversal methods maintain much higher graph edge ratios.

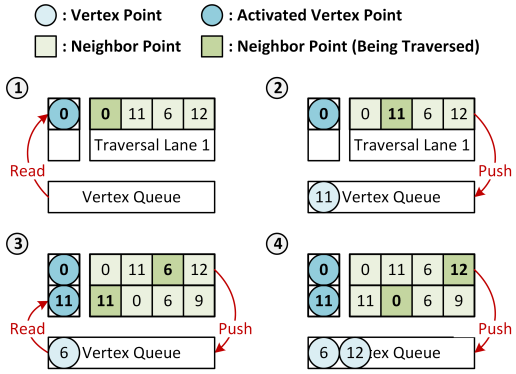


**Figure 5: Illustration of SBFS with 2 traversal lanes.**

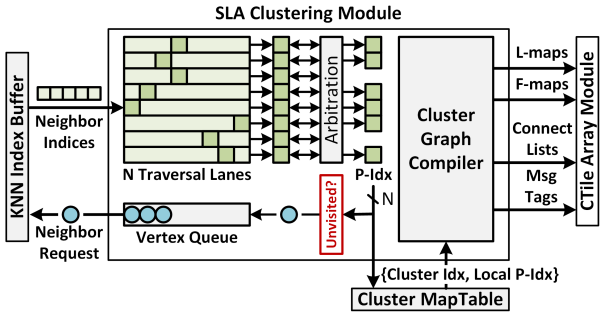


**Figure 6: Architecture of SLA clustering module.**

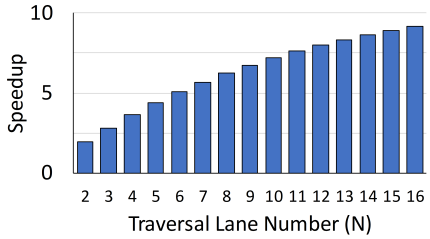


**Figure 7: Speedup of the SLA clustering module using SBFS with $N$ traversal lanes over the BFS baseline.**

The improvement over the baseline reaches 7.9 to 10.5× at 10k points. All traversal methods also outperformed the baseline in graph edge length as shown in Figure 4(b). Compared to the baseline at 1k points, BFS, DFS, and BDFS reduce the graph edge length by 4.7×, 1.4×, and 2×, respectively. For larger point sizes, the baseline and depth-first methods incur significant increases in graph edge lengths, whereas BFS shows only limited increase. BFS reduces the graph edge length by 14× over the baseline at 10k points. Combining both fine-grained and coarse-grained spatial locality, BFS stands out as the most promising method, especially notable for its scalable graph edge length.

## 3.2 SBFS Traversal

To ensure correctness, BFS only allows traversing neighbors of one vertex point at a time. This requirement hinders the parallelization of BFS traversal execution [23]. We propose the speculative BFS (SBFS) algorithm to approximate BFS traversal and parallelize execution by speculating the traversal order. The speculation is possible thanks to the community structure of a KNN graph. If two points are connected, they are also likely to share neighbors. For instance, in Figure 3(a), point 0 and point 11 are connected and share common neighbors of points 0, 11, and 6. Traversing these two points in parallel does not affect the traversal order.

Figure 5 illustrates the first four iterations of traversing the graph in Figure 3(a) using an SBFS-2 algorithm, where 2 indicates two active lanes for vertex traversal. Untraversed vertices are read from the vertex queue and its neighbor points are loaded to a traversal lane for traversal. Here is a step-by-step rundown of the process:

(1) The root point 0 is read from the vertex queue and its neighbors 0, 11, 6, 12 are loaded into traversal lane 0.
(2) From traversal lane 0, neighbor point 11 is traversed. Since point 11 is not visited yet, it is pushed into the vertex queue.
(3) From traversal lane 0, neighbor point 6 is traversed and pushed into the vertex queue. In the meantime, vertex point 11 is read from the vertex queue and its neighbors 11, 0, 6, 9 are loaded into traversal lane 1.
(4) From traversal lane 0, point 12 is traversed and pushed into the vertex queue, and traversal lane 0 is cleared. From traversal lane 1, neighbor point 0 is traversed. Since point 0 is already visited, it is skipped.

By traversing the neighbors of multiple neighboring vertices at a time, the execution time can be largely reduced with only minor changes to the graph traversal order. The clustering metrics of SBFS-16 are plotted in Figure 4. SBFS-16 produces a similar graph edge ratio and graph edge length as BFS. The speculation produces a close approximation to BFS in terms of clustering quality.

## 3.3 SLA Clustering Module Implementation

As Figure 6 shows, the SLA clustering module is realized by an SBFS traversal module and a cluster graph compiler module for converting a global graph into local subgraphs for each cluster and setting up the inter-cluster connections.

The KNN index buffer stores the constructed KNN graph in the adjacency list format. The SBFS module reads an untraversed vertex point from the vertex queue, and the neighbor indices are requested from the KNN buffer for loading to the traversal lanes. The SBFS module consists of $N$ traversal lanes operating in parallel. Consecutively traversed neighbor points are recorded in the cluster maptable. The cluster maptable stores the mapping between the point index and the corresponding pair of {cluster index, local point index}. If a point index is not already recorded in the maptable, a new {cluster index, local point index} pair is assigned in ascending order and written to the maptable. Based on the outputs of the SBFS module and the maptable, the cluster graph compiler module generates subgraphs and inter-cluster connections in the form of local maps (L-maps) and foreign maps (F-maps) (see usage in Section 5). For inter-CTile data exchange, the cluster graph compiler generates the connectivity list and message tags (see usage in Section 4).

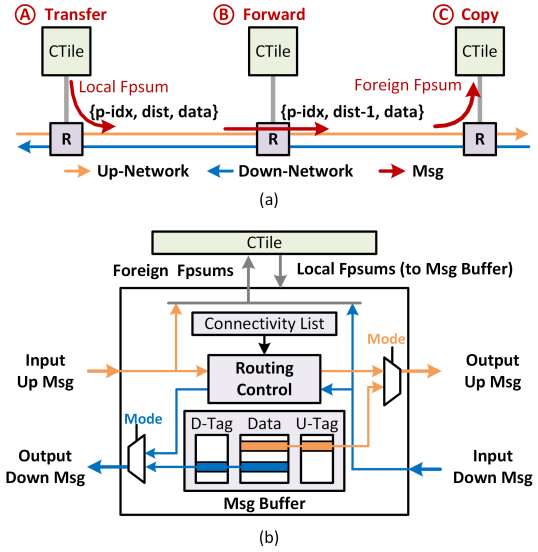


**Figure 8: Architecture of (a) a chain NoC, and (b) a router for chain NoC.**

The SBFS-based SLA clustering module speeds up clustering as show in Figure 7. For example, SBFS-16 reduces the traversal latency by 9.2×. With more traversal lanes, nearly linear speedup can be obtained, but the speedup slowly diminishes due to arbitration and bandwidth limitation of the vertex queue.

# 4 LOCALITY-AWARE NOC

The inter-CTile data exchange is supported by a NoC. A general-purpose mesh NoC forwards data in four directions at each router node to allow flexible routing. In comparison, we propose a locality-aware chain NoC that takes advantage of the extracted spatial locality to achieve a lower complexity and a higher efficiency. The chain NoC is designed with a message reuse strategy to further improve performance and reduce area and power.

## 4.1 Chain NoC Architecture

A chain NoC connects routers using two independent uni-directional networks: an up network transfers messages upward, whereas a down network transfers messages downward as shown in Figure 8. A message comprises a data field and a tag: the data is the neighbor fpsum computed by the source CTile and the tag consists of the point index and the directive for routing the message to its destination.

The chain NoC is designed with a message reuse strategy where a message transfer passes through all its destination CTiles in the same direction to save redundant transfers. In SLA clustering, the cluster graph compiler (Figure 6) keeps track of the furthest destination CTile that an fpsum needs to travel to in both upward and downward directions. This approach simplifies the routing directive of a message tag into a single distance value between the source and the furthest destination. By adopting the message reuse strategy, the chain NoC cuts the message traffic by 2.1×, which contributes to a higher efficiency for inter-CTile data exchange.

**Algorithm 1:** Chain NoC Routing

```
Input/Output: msg_in/msg_out
if (msg_in != None) then // Receive msg_in
    {p-idx, dist, data} = msg_in
    if (dist > 1) then // Keep forwarding msg_in
        Forward: msg_out = {p-idx, dist-1, data}
    else // msg_in reaches destination, free to transfer
        Read {p-idx', dist', data'} from Msg Buffer
        Transfer: msg_out = {p-idx', dist', data'}
    if (p-idx match in connectivity list) then
        Copy: send data to CTile
else // No msg_in, free to transfer
    Read {p-idx', dist', data'} from Msg Buffer
    Transfer: msg_out = {p-idx', dist', data'}
```

## 4.2 Routing Algorithm

The routing mechanism of the chain NoC is illustrated in Figure 8(a). In preparation for exchanges, the router associated with a CTile receives the connectivity list and the message tags from the SLA clustering module, and the local neighbor fpsums from the CTile. The message tags and local neighbor fpsums are stored in the message buffer. The connectivity list contains the indices of neighbor fpsums that need to be fetched from other CTiles.

An fpsum exchange undergoes three stages as shown in Figure 8(a): in the transfer mode, a local router reads a pair of tag and data from the message buffer and sends a message onto the network; in the forward mode, routers forward the message along the way; and when an incoming message tag matches the connectivity list of a destination router, the message data is copied to the destination CTile in the copy mode. The routing algorithm is detailed in Algorithm 1. A message is propagated along the up or down direction. The message distance is reduced by 1 when passing through a hop in the forward mode until the message reaches its final destination. By prioritizing the forward mode over the transfer mode, a message is never stalled before reaching its final destination.

Compared to a flexible mesh NoC, the uni-directional design of the chain NoC reduces the complexity and energy spent on complex message switching and arbitration. Furthermore, the routing algorithm ensures that a message is never stalled when traveling in the network, thus eliminating extra input and output buffers in the router design. As a result, the chain NoC has a low design complexity and incurs minimal area and energy overheads. In terms of performance, the chain NoC provides a better fetch efficiency than the mesh NoC even though the chain NoC uses only half of the mesh NoC's physical bandwidth. By message reuse, the chain NoC reduces the transfer latency by 2.1× over the mesh NoC. When SLA clustering is applied to both types of NoCs, the chain NoC reduces the transfer latency by 3.2× over the mesh NoC, demonstrating its suitability for moving data with spatial locality.

# 5 CTILE ARCHITECTURE

A CTile supports the operations required for EdgeCONV, including KNN graph construction and GraphCONV. KNN graph construction is supported by DP for distance computation and K-min sorting; and GraphCONV is supported by DP for fpsum computation and feature

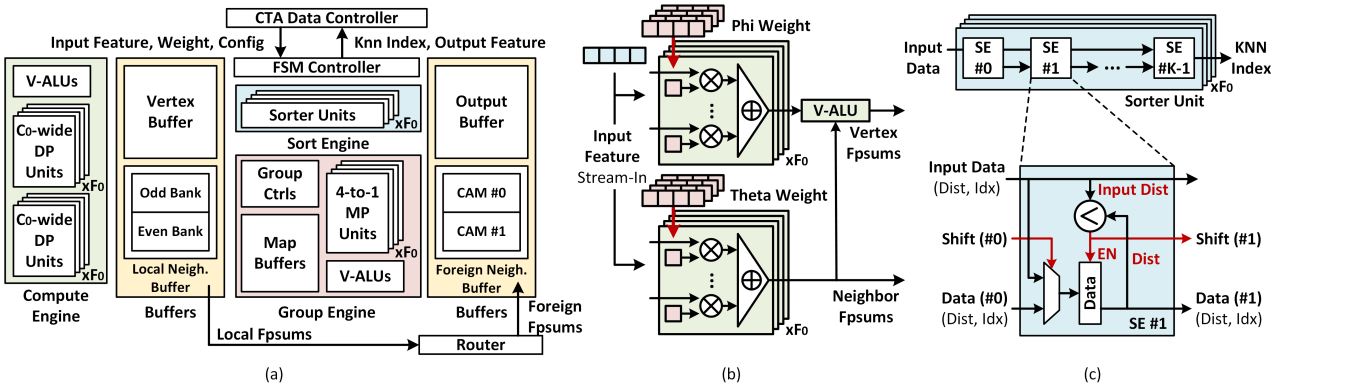


**Figure 9: Microarchitecture of (a) a compute tile (CTile), (b) a compute engine, and (c) a sort engine.**

aggregation. A CTile shown in Figure 9(a) comprises of a compute engine, a sort engine, and a group engine for feature aggregation. It also consists of buffers for storing fpsums and a controller for interfaces and configurations. Finally, an output buffer stores the results of CTile computation.

***Fpsum Computation.*** The compute engine contains 2 sets of DP units, $F_0$ phi-DP units and $F_0$ theta-DP units ($F_0$ is a design parameter), and vector arithmetic units (V-ALUs) as shown in Figure 9(b). A DP unit caches and reuses the respective $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ weights while the inputs are streamed in to compute DP. A DP unit computes a $C_0$-way DP per cycle ($C_0$ is another design parameter). Altogether, $F_0$ neighbor fpsums are computed per cycle by the theta-DP units, and $F_0$ vertex fpsums are computed per cycle by the subtraction of the outputs of the theta-DP units from the phi-DP units. The vertex and neighbor fpsums are stored in vertex and local neighbor buffers for reuse.

***K-Min Sorting for KNN.*** The sort engine contains $F_0$ K-min sorter units as illustrated in Figure 9(c). A sorter unit finds the KNN of one given vertex point using insertion sort that is implemented in a 1D systolic array of $K$ sorting elements (SEs) [50]. A distance and point index pair is broadcast to all SEs. Each SE compares the distance value to its current value, inserts the new value or takes the shifted value from the left SE. A K-min sorter unit always maintains the $K$ nearest points of a vertex point sorted by distance values.

***Feature Aggregation.*** The group engine shown in Figure 10 performs feature aggregation including max-pool (MP) of the neighbor fpsums, summing the max neighbor fpsum to the vertex fpsum, and applying ReLU as described by Eq. (3).

The MP process is of particular importance as it needs to access neighbor fpsums that reside locally and on foreign CTiles. The local controller uses L-maps provided by the SLA clustering module to locate local neighbor fpsums. The L-map of a vertex point stores the local connections to the vertex point in adjacency matrix form. An L-map word is decoded into point indices for accessing the neighbor fpsums stored in the local buffer. To speed up fetching, the local neighbor buffer can be implemented using two 2-port SRAM banks. Two even and two odd point indices are decoded from an L-map every cycle to allow 4 neighbor fpsum words to be

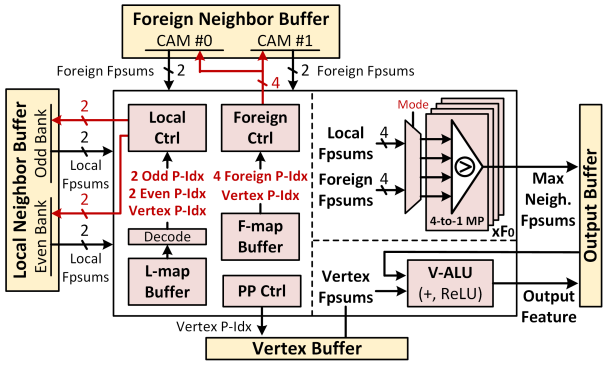


**Figure 10: Microarchitecture of a group engine.**

fetched per cycle. Each fpsum word contains $F_0$ values. The four fpsum words are sent to the MP units.

The foreign controller uses F-maps provided by the SLA clustering module to locate neighbor fpsums from foreign CTiles. An F-map of a vertex point contains its foreign neighbor point indices. An F-map word contains 4 point indices. The foreign controller reads one word from an F-map per cycle, decodes the 4 point indices, and looks up the indices in two content-addressable memories (CAMs). Each CAM returns at most 2 neighbor fpsum words. If a CAM has more than 2 matches, the foreign controller stalls for a cycle before proceeding to the next word. Up to 4 neighbor fpsum words are fetched per cycle and sent to the MP units.

Once all local and foreign neighbor fpsums are aggregated to obtain the max neighbor fpsum, it is summed with the vertex fpsum, followed by ReLU to produce the output feature.

## 6 POINT-X SYSTEM ARCHITECTURE

The Point-X system architecture is outlined in Figure 11. Point-X consists of a clustering module, a CTile array (CTA) module, data buffers, and controllers for system configuration, dataflow, and external communication. The system controller receives the compiled instructions from the host processor, and configures the other modules according to the dataflow mode. The memory controller loads

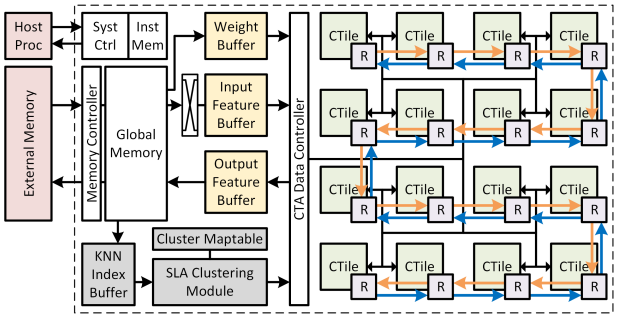


**Figure 11: Point-X system architecture.**

the input features, model weights, and KNN indices, and offloads the output features to the external memory. The global memory holds the data fetched from off-chip and distributes them to respective data buffers following the dataflow mode. The KNN index buffer stores the constructed KNN graph, and the cluster maptable stores the mapping after SLA clustering.

The clustering module performs SLA clustering and compiles the clustering configuration for the CTA module. In the prototype design, the CTA module consists a 4×4 2D array of CTiles connected by a chain NoC with upward and downward networks. A data controller handles the input distribution to the CTiles using a pipelined H-tree bus. Inside a CTile, a compute engine contains 16 8-way 16b×8b DP units ($F_0$ = 8, $C_0$ = 8), reconfigurable to 8 8-way 16b×16b DP units. A group engine has 8 4-to-1 MP units. A sort engine has 8 K-min sorter units with 20 SEs ($K$ = 20) each. Overall, the Point-X prototype contains 16 CTiles, using 2,048 16b×8b MACs (1,024 16b×16b MACs) in total.

## 6.1 Multi-Mode Dataflow

Typical graph-based DNNs are composed of EdgeCONV (KNN graph construction and GraphCONV), shared MLP, and FC layers with different computational requirements. Point-X provides a multi-mode dataflow by coordinating the input loading and data distribution to support these diverse requirements. The dataflow for EdgeCONV, shared MLP and FC layers are illustrated in Figure 12.

***KNN Graph Construction*** *(Figure 12(a)).* Vertex point features are loaded to DP units of each CTile by unicast and the input point features are streamed in to the CTiles by broadcast. A pair of 16b×8b DP units are combined to compute 16b-16b point feature distances: one DP unit for the MSB part and the other for the LSB part. A V-ALU performs shift and accumulation of the two psums. The distance values and point indices are sent to the sort engine (Figure 9(c)). At completion, the $K$ indices in each sorter unit are read out.

***GraphCONV*** *(Figure 12(c)).* The SLA clustering module loads point indices from the KNN index buffer, performs SBFS traversal to produce the clustering mapping, and compiles cluster configurations, including L-maps, F-maps, connectivity lists, and message tags.

Following the cluster maptable, the input features are stored in the corresponding cluster bank. The weights are broadcast to all CTiles, and the input features are unicast to the corresponding

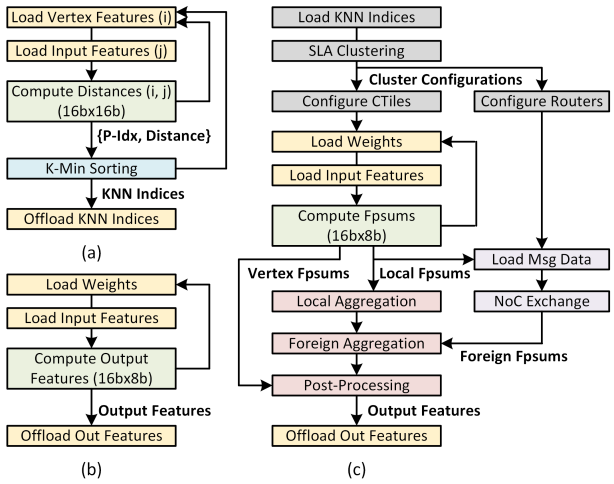


**Figure 12: Multi-mode dataflow for (a) KNN graph construction, (b) shared MLP and FC, and (c) GraphCONV operations.**

CTiles. The vertex and neighbor fpsums are computed and stored in the local buffers. NoC data exchange can be executed concurrently with the local fpsum aggregation. The foreign neighbor fpsums are received over the NoC, and the group engine performs the foreign fpsum aggregation. Lastly, post-processing (MP, summing and ReLU) is performed.

***Shared MLP and FC Layers*** *(Figure 12(b)).* The weights are broadcast to the CTiles, and the input features are streamed in and unicast to their corresponding CTiles. In the FC mode, each CTile receives 2 weights via unicast and the input vector is broadcast to the CTiles for computation.

## 6.2 Workload Partitioning

The Point-X prototype is designed for processing up to 1k points from the input point cloud at a time. Point clouds of size of over 1k points are partitioned into 1k blocks for Point-X to process.

For KNN graph construction, shared MLP, and FC operations, an input point cloud is sequentially partitioned into 1k-blocks and processed with corresponding inputs or weights. For GraphCONV, the sequential partition is not the most efficient. As shown in Figure 13(a), the sequential partition divides the adjacency matrix in sequentially-ordered blocks. Although being simple to execute, as the point size scales up, the scheme results in quadratically increasing number of blocks to fetch.

In Point-X, we adopt a diagonal partition by exploiting the SLA clustering as illustrated in Figure 13(b). The SLA clustering is applied to the entire input point cloud. Only the diagonal blocks in the adjacency matrix are sent to Point-X for processing. As discussed in Section 3, the SLA clustering increases the spatial locality and computation efficiency by increasing the number of local edges and bringing the foreign edges closer to the matrix diagonal. Consequently, the diagonal partition provides two major advantages. First, the number of blocks scales linearly with the point size. Only a small number of cross-block foreign edges need to be handled.

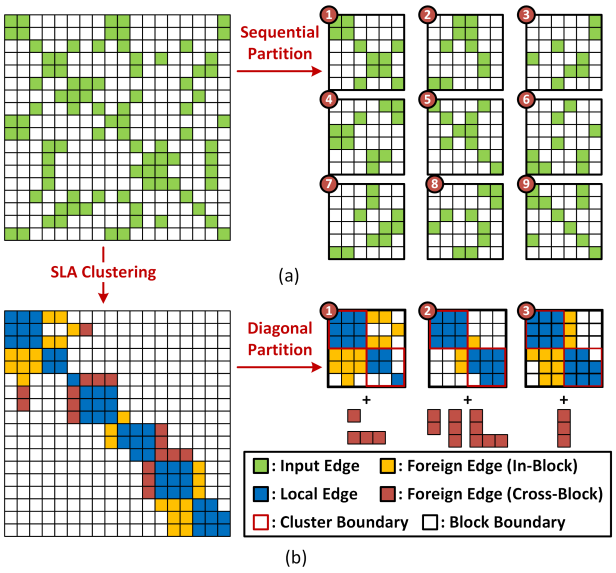


**Figure 13: Workload partition schemes: (a) sequential partition, (b) diagonal partition.**

Second, the diagonal partition provides more spatial locality per block, resulting in a higher computation efficiency.

In handling large point clouds, the SLA clustering module needs to fetch KNN indices of vertex points scattered in off-chip memory, which can cause a significant delay. To reduce the delay, the global memory is used to prefetch the KNN indices to supplement the KNN index buffer. Following the completion of clustering, Point-X fetches points of diagonal blocks from the off-chip memory. To improve the point fetch efficiency, the clustering results are organized so that the points located in the same DRAM page are accessed together. This approach helps to avoid unnecessary DRAM page changes and provides a burst-like access bandwidth.

## 7 BENCHMARKING AND EVALUATION

A prototype of Point-X is designed in a 28nm CMOS technology to evaluate its performance, area, and efficiency. The results are compared to a CPU, a GPU, accelerator baselines, and state-of-the-art accelerator works [41, 24] to show Point-X's advantages in throughput and energy efficiency in processing graph-based point-cloud DNNs.

### 7.1 Evaluation Methodology

Following [33, 35, 48, 60, 6, 47, 56, 25, 53, 26, 3, 44, 5, 41, 24], we used the sampled point cloud dataset from ModelNet40 [51] with a point cloud size of 1,024. Quantization-aware training was applied to reduce the precision of the input features and weights to 16b and 8b, respectively. The higher input feature precision is necessary for a larger point cloud size. For instance, an 8b input precision allows only 256 distinct positions in each dimension for the 1,024 points, making points indistinguishable. The 16b-input, 8b-weight DGCNN achieves 92.8% accuracy on ModelNet40, and shows no accuracy drop compared to the DGCNN trained in FP32 [48]. To evaluate

**Table 2: Storage and Area Breakdown of Point-X**

| Module | Storage (KB) | Area ($mm^2$) | Area Ratio (%) |
|---|---|---|---|
| CTile | 18.39 | 0.288 | 4.25 |
| Router | 2.09 | 0.023 | 0.34 |
| CTile Array (CTA) | 327.75 | 4.935 | 72.90 |
| SLA Clustering | 26.88 | 0.566 | 8.36 |
| System Ctrl & Mem | 190.75 | 1.270 | 18.76 |
| Point-X Total | 545.4 | 6.8 | 100 |

**Table 3: Layer Evaluation of Point-X on 1k-DGCNN [48]**

| Layer (Kernel Size, $F \times C$) | | Latency ($\mu s$) (Overhead) | Energy ($\mu J$) (Overhead) |
|---|---|---|---|
| EdgeCONV-1 (64×3) | KNN | 8.3 | 17.0 |
| | GraphCONV | 10.2 (25.9%) | 7.5 (6.5%) |
| EdgeCONV-2 (64×64) | KNN | 74.6 | 158.5 |
| | GraphCONV | 16.8 (15.7%) | 18.3 (2.6%) |
| EdgeCONV-3 (128×64) | KNN | 74.6 | 158.5 |
| | GraphCONV | 31.0 (8.5%) | 35.9 (1.3%) |
| EdgeCONV-4 (256×128) | KNN | 148.0 | 316.7 |
| | GraphCONV | 89.8 (2.9%) | 136.1 (0.4%) |
| Shared MLP (1024×512) | | 307.2 | 799.1 |
| FC-1 (512×1024) | | 3.2 | 5.2 |
| FC-2 (256×512) | | 0.9 | 1.3 |
| FC-3 (40×256) | | 0.2 | 0.2 |
| DGCNN Total | | 765.0 (1.38%) | 1654.3 (0.12%) |

Point-X's performance for scaled-up point cloud data, augmented datasets with 2k, 4k, 6k, 8k, and 10k points are generated by re-sampling the original CAD models in ModelNet40. The augmented datasets represent point clouds of higher resolutions.

A cycle-accurate model was developed to simulate the behavior and analyze the performance of Point-X's architecture and dataflow. For benchmarking, cycle-accurate models were also implemented for a query-based baseline and an exchange-based baseline.

A prototype of Point-X was implemented in RTL and synthesized in a 28nm CMOS technology with SRAMs generated from SRAM compilers to provide more accurate silicon area, timing, and power estimates. PrimeTime PX was used to estimate the energy consumption of Point-X based on activity waveforms in running complete network layers of our workloads.

For performance comparison, we evaluated the query-based and exchange-based baselines using the same workloads. Both baselines were designed using the same number of CTiles, memory banks, and MACs in each DP unit as Point-X, but differ from Point-X in the data fetch mechanism. The exchange-based baseline is designed with a mesh NoC for data exchange between CTiles. Lastly, we compare Point-X's efficiency in EdgeCONV to existing accelerators [41, 24] and Point-X's end-to-end throughput and efficiency to an Nvidia GTX-1080Ti GPU and an Intel i7-7700K CPU.

### 7.2 Area, Performance, Efficiency Analysis

The memory and silicon area breakdown of the Point-X prototype are shown in Table 2. The Point-X prototype is 6.8 $mm^2$ in size and

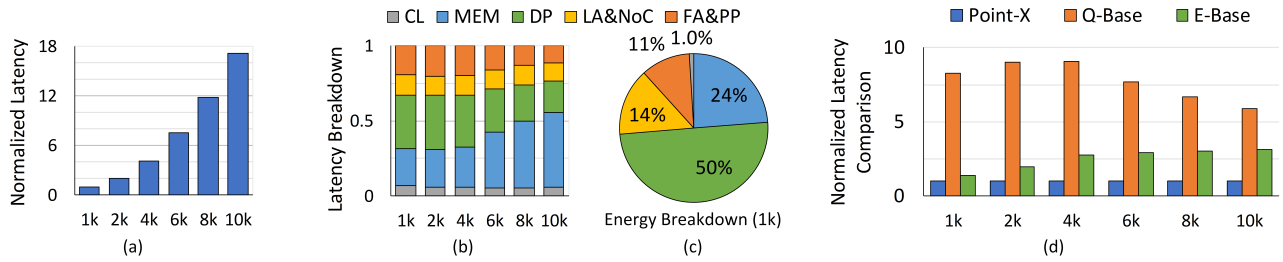


**Figure 14: (a) Normalized latency, (b) latency breakdown, and (c) energy breakdown of Point-X for GraphCONV in DGCNN; (d) normalized latency comparison of Point-X to Q-Base and E-Base baselines for GraphCONV for different point cloud sizes.**

contains 545.4 KB of on-chip memory. The SLA clustering module costs an area overhead of only 8.36%. Inside the CTA module, a CTile is 0.29 $mm^2$, while a chain NoC router is only 0.023 $mm^2$, which is less than 1/10 the size of a CTile.

The layer-by-layer latency and energy results of the Point-X prototype when running 1k-DGCNN are presented in Table 3. The results for each EdgeCONV layer are shown separately for KNN graph construction and GraphCONV. For every GraphCONV, the overhead of SLA clustering is noted in parentheses. For a small EdgeCONV layer like EdgeCONV-1, the SLA clustering costs an overhead of 25.9% in latency and 6.5% in energy. The SLA clustering overhead decreases in larger EdgeCONV layers, e.g., it takes only 2.9% and 0.4% of the latency and energy in EdgeCONV-4. For the 1k-DGCNN workload, the Point-X prototype demonstrates an end-to-end latency of 0.77 ms at an average power consumption of 2.2 W. It achieves a throughput of 1307.1 Inf./s at 604.5 Inf./J.

## 7.3 Workload Scalability Analysis

Point-X is evaluated for larger workloads. Figure 14(a) and (b) show the normalized latency and the latency breakdown of Point-X when running GraphCONV in DGCNN for different point cloud sizes. The latency breakdown looks at the five components of GraphCONV: SLA clustering (CL), fpsum compute (DP), local fpsum aggregation and NoC data exchange (LA&NoC), and foreign fpsum aggregation and post-processing (FA&PP), and system control and memory (MEM). The MEM component includes the latency of configuration and data fetch that cannot be fully hidden by prefetching.

For a point size of under 4k, the global memory serves both the clustering module and the CTA module effectively by prefetching the inputs needed for GraphCONV. This can be observed by the almost linear increase in the normalized latency and consistent latency breakdown. For a point size larger than 4k, the input points needed for processing a block in GraphCONV may be located across multiple DRAM pages, resulting in an increase in MEM latency as shown in Figure 14(b). The energy breakdown of Point-X for the GraphCONV workloads is shown in Figure 14(c).

## 7.4 Performance Comparison

In Figure 14(d), we show Point-X's improvement over a query-based design (Q-Base) and an exchange-based design (E-Base). The differences between Point-X and the two base designs are mainly attributed to the fetch mechanisms and the workload partition schemes for GraphCONV. In Q-Base, CTiles request neighbor points from a centralized shared memory. It incurs a low fetch efficiency due to memory access conflicts. In E-Base, CTiles communicate with each other through a mesh NoC to obtain the needed neighbor points. E-Base suffers from the overloading of message traffic across the network and the long latency of message transfer, which worsen its fetch efficiency. Overall, Point-X provides a speedup of 8.3× and 1.4× over Q-Base and E-Base, respectively, for GraphCONV workloads of 1k points.

The benefit of Point-X's diagonal workload partition is illustrated in Figure 14(d) showing normalized comparison for each point cloud size. As the point size increases, the efficiency of Point-X's point fetching from DRAM decreases. As a result, the advantage of Point-X over Q-Base shrinks slightly due to the reduced number of edges per block (for a fixed $K$) and less frequent memory access conflicts that favor Q-Base. Point-X achieves an average speedup of 7.7× and 2.5× in GraphCONV computation across different point sizes compared to Q-Base and E-Base, respectively.

Table 4 compares Point-X to state-of-the-art accelerators for EdgeCONV computation [41, 24]. Cambricon-G [41] has the same peak throughput as Point-X but it uses 22× on-chip memory storage. The large storage holds all intermediate results on-chip for reuse and reduces the latency and energy overheads from external memory communication. DeepBurning-GL [24] is implemented on an FPGA using a customized EdgeCONV template. Both Cambricon-G and DeepBurning-GL process EdgeCONV following Eq. (2) and they do not exploit the locality in point cloud, resulting in less-efficient data fetching and redundant computation. Overall, Point-X achieves

**Table 4: EdgeCONV Comparison to Existing Works**

| | Point-X | Cambricon-G | DeepBurning-GL |
|---|---|---|---|
| Type | ASIC | ASIC | FPGA |
| Technology | 28 nm | 45 nm | 16 nm |
| Frequency | 1.0 GHz | 1.0 GHz | 200 MHz |
| Compute Unit | 2048 MACs | 2048 MACs | 5893 DSPs[c] |
| Precision | 16b & 16b/8b | 16b | 16b |
| On-Chip Mem. | 545.4 KB | 12.1 MB | 3.2 MB[c] |
| Efficiency (GOPS/W) | 858.6[a] | 360.9[b] | 71.1[b] |

[a] A MAC is counted as 2 OPs; comparisons in KNN are included for energy but not counted in total OPs; Average from all EdgeCONV in [48].
[b] Numbers derived from [41, 24]; Only EdgeCONV (64×3) was reported.
[c] Derived from resource utilization reported in [24] for Alveo U50 FPGA.

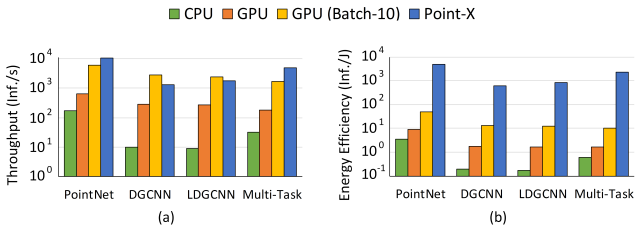


**Figure 15: Comparison of (a) throughput and (b) energy efficiency of Point-X to the CPU and GPU baselines.**

a 2.4× and 12.1× higher efficiency in EdgeCONV computation over Cambricon-G and DeepBurning-GL, respectively.

Point-X is also compared to a GPU (Nvidia GTX-1080Ti) and a CPU (Intel i7-7700k). Both the GPU and the CPU are in more advanced, faster, and more efficient silicon technologies than the 28nm used for Point-X prototyping. The GPU and the CPU run at higher clock frequencies, have larger silicon footprints, and contain larger on-chip memories for caching. In contrast, Point-X is designed for graph-based DNNs. It has a much smaller silicon footprint and consumes less power. Point-X provides a total of 2,048 16b×8b MACs distributed to 16 CTiles, whereas the GPU has 3,584 cores and the CPU has 8 threads for floating point computation. Figure 15 compares the throughput and energy efficiency of Point-X to the GPU and CPU baselines for four workloads. When running DGCNN, Point-X achieves a 4.5× and a 129.7× higher throughput over the GPU and the CPU, respectively, at a 342.9× and a 3160.9× better energy efficiency, respectively. We also compared to the GPU with batch-10 inference. On average, the GPU throughput is improved by almost 10× with only a 35% power increase. However, batching is often infeasible for devices with limited memory or real-time computing use cases.

## 8 RELATED WORK

Graph-based point-cloud DNNs possess both regular computation structures that require plenty of data reuse and parallelism, and irregular computation structures that require flexible and efficient data fetch mechanisms. DNN/CNN accelerators [16, 8, 38] are designed for regular computation structures, and they tend to perform poorly on graph-structured data featuring scattered memory access and limited data reuse. On the other hand, graph processing accelerators [13, 31] are designed for sparse and irregular data access and computation, and they are unable to fully exploit the parallelism and data reuse in DNN-like workloads.

GNN/GCN accelerators [54, 18, 11, 7, 41, 24] address both computation structures. These accelerators work on arbitrary graphs with power-law distribution, which presents severe workload imbalance in processing. HyGCN [54] and GRIP [18] proposed window sliding/shrinking and parallel prefetching methods to improve fetch efficiency, whereas AWB-GCN [11] proposed runtime workload rebalancing methods to improve compute utilization. These works focus on static graphs and are unable to fully support EdgeCONV due to the lack of runtime graph construction units. Recently, Cambricon-G [41] and DeepBurning-GL [24] were proposed to support dynamic graphs, i.e., EdgeCONV, but they do not exploit the data locality in point cloud data, making them unable to provide the best efficiency. In comparison, Point-X leverages data reuse opportunities to eliminate redundant computation and exploits spatial locality to achieve a higher computational efficiency.

Rubik [7] and GNNAdvisor [46] proposed graph reordering techniques using locality-sensitive hashing and Rabbit Order [2] to increase locality during GNN processing. The input graph is first preprocessed offline before being sent to the GPU or accelerator for processing. For an EdgeCONV operation where the graph is constructed in runtime, offline preprocessing is impractical due to data transfer overheads between the host and the accelerator. In contrast, Point-X avoids the data transfer by having an on-chip clustering implementation which can provide a significant speedup over software implementations in [7, 46].

Low-cost NoC router designs were explored in the past [17, 32]. The routing is limited to x- or y-directions in [17] to reduce the design complexity. Both [17, 32] prioritized in-network traffic to reduce the buffers needed in a router. Similarly, Point-X reduces the 2D mesh to 1D chain to avoid the switching overheads and prioritizes the forward mode to eliminate buffers in routers. Accelerators [9, 21, 20] also adopted NoCs for inter-PE or memory-to-PE communication, allowing more flexibility to support diverse workloads and dataflows. In comparison, Point-X's NoC is specialized for efficient data exchange with coarse-grained spatial locality and targets both compact area and stringent power budgets.

## 9 CONCLUSION

We present Point-X, an SLA accelerator architecture that extracts and leverages spatial locality for efficient graph-based DNN processing on point clouds. To extract spatial locality in point clouds, an SBFS graph traversal algorithm is presented to map points into clusters to increase intra-CTile computation parallelism and reduce the inter-CTile communication overhead. Compared to conventional graph traversal methods, SBFS can be parallelized to achieve a 9.2× speedup. To leverage the spatial locality, a lightweight chain NoC is presented to reduce the data exchange latency by 3.2× compared to a mesh NoC. Combining these innovations, Point-X is designed with a multi-mode dataflow to support all operations in graph-based DNNs, i.e., EdgeCONV, shared MLP, and FC layers. A 1.0 GHz Point-X prototype is designed in a 28nm technology, occupying 6.8 $mm^2$ of silicon area. Point-X achieves up to 12.1× higher energy efficiency in EdgeCONV compared to state-of-the-art accelerators. When evaluated on the DGCNN workload, Point-X achieves a throughput of 1307.1 Inf./s and an energy efficiency of 604.5 Inf./J. Compared to an Nvidia GTX-1080Ti GPU, Point-X demonstrates a 4.5× and a 342.9× improvement in throughput and energy efficiency, respectively, on DGCNN workload.

## ACKNOWLEDGMENTS

The authors would like to thank Yi-Chung Wu, Reid Pinkham, Shang-En Huang, Chester Liu, and Wei Tang for the valuable discussions and help.

## REFERENCES

[1] Eman Ahmed, Alexandre Saint, Abd El Rahman Shabayek, Kseniya Cherenkova, Rig Das, Gleb Gusev, Djamila Aouada, and Björn E. Ottersten. 2018. Deep Learning Advances on Different 3D Data Representations: A Survey. arXiv:1808.01462

[2] Junya Arai, Hiroaki Shiokawa, Takeshi Yamamuro, Makoto Onizuka, and Sotetsu Iwamura. 2016. Rabbit Order: Just-in-Time Parallel Reordering for Fast Graph Analysis. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. 22–31.

[3] Matan Atzmon, Haggai Maron, and Yaron Lipman. 2018. Point Convolutional Neural Networks by Extension Operators. *ACM Transactions on Graphics* 37, 4, Article 71 (July 2018).

[4] Jay Banerjee, Won Kim, S.-J. Kim, and Jorge F. Garza. 1988. Clustering a DAG for CAD Databases. *IEEE Transactions on Software Engineering* 14, 11 (Nov. 1988), 1684–1699.

[5] Can Chen, Luca Zanotti Fragonara, and Antonios Tsourdos. 2019. GAPNet: Graph Attention based Point Neural Network for Exploiting Local Feature of Point Cloud. arXiv:1905.08705

[6] Chao Chen, Guanbin Li, Ruijia Xu, Tianshui Chen, Meng Wang, and Liang Lin. 2019. ClusterNet: Deep Hierarchical Cluster Network With Rigorously Rotation-Invariant Representation for Point Cloud Analysis. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 4989–4997.

[7] Xiaobing Chen, Yuke Wang, Xinfeng Xie, Xing Hu, Abanti Basak, Ling Liang, Mingyu Yan, Lei Deng, Yufei Ding, Zidong Du, Yunji Chen, and Yuan Xie. 2020. Rubik: A Hierarchical Architecture for Efficient Graph Learning. arXiv:2009.12495

[8] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (Jan. 2017), 127–138.

[9] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (June 2019), 292–308.

[10] Raveesh Garg, Eric Qin, Francisco Muñoz Martínez, Robert Guirado, Akshay Jain, Sergi Abadal, José L Abellán, Manuel E. Acacio, Eduard Alarcón, Sivasankaran Rajamanickam, and Tushar Krishna. 2021. A Taxonomy for Classification and Comparison of Dataflows for GNN Accelerators. arXiv:2103.07977

[11] Tong Geng, Ang Li, Runbin Shi, Chunshu Wu, Tianqi Wang, Yanfei Li, Pouya Haghi, Antonino Tumeo, Shuai Che, Steve Reinhardt, and Martin C. Herbordt. 2020. AWB-GCN: A Graph Convolutional Network Accelerator with Runtime Workload Rebalancing. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 922–936.

[12] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. 2020. Deep Learning for 3D Point Clouds: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (Early Access)* (2020). https://doi.org/10.1109/TPAMI.2020.3005434

[13] Tae Jun Ham, Lisa Wu, Narayanan Sundaram, Nadathur Satish, and Margaret Martonosi. 2016. Graphicionado: A High-Performance and Energy-Efficient Accelerator for Graph Analytics. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 1–13.

[14] Kaveh Hassani and Mike Haley. 2019. Unsupervised Multi-Task Feature Learning on Point Clouds. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 8159–8170.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[16] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 1–12.

[17] John Kim. 2009. Low-Cost Router Microarchitecture for On-Chip Networks. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 255–266.

[18] Kevin Kiningham, Christopher Re, and Philip Levis. 2020. GRIP: A Graph Neural Network Accelerator Architecture. arXiv:2007.13828

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. 1097–1105.

[20] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2017. Rethinking NoCs for Spatial Neural Network Accelerators. In *International Symposium on Networks-on-Chip (NOCS)*. 1–8.

[21] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS)*. 461–475.

[22] Shiyi Lan, Ruichi Yu, Gang Yu, and Larry S. Davis. 2019. Modeling Local Geometric Structure of 3D Point Clouds Using Geo-CNN. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 998–1008.

[23] Charles E. Leiserson and Tao B. Schardl. 2010. A Work-Efficient Parallel Breadth-First Search Algorithm (or How to Cope with the Nondeterminism of Reducers). In *Proceedings of the Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 303–314.

[24] Shengwen Liang, Cheng Liu, Ying Wang, Huawei Li, and Xiaowei Li. 2020. DeepBurning-GL: An Automated Framework for Generating Graph Neural Network Accelerators. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. Article 72.

[25] Yongcheng Liu, Bin Fan, Gaofeng Meng, Jiwen Lu, Shiming Xiang, and Chunhong Pan. 2019. DensePoint: Learning Densely Contextual Representation for Efficient Point Cloud Processing. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 5238–5247.

[26] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. 2019. Relation-Shape Convolutional Neural Network for Point Cloud Analysis. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 8887–8896.

[27] Daniel Maturana and Sebastian Scherer. 2015. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*. 922–928.

[28] Bert Moons, Roel Uytterhoeven, Wim Dehaene, and Marian Verhelst. 2017. Envision: A 0.26-to-10TOPS/W Subword-Parallel Dynamic-Voltage-Accuracy-Frequency-Scalable Convolutional Neural Network Processor in 28nm FDSOI. In *Proceedings of the International Solid-State Circuits Conference (ISSCC)*. 246–247.

[29] Anurag Mukkara, Nathan Beckmann, Maleen Abeydeera, Xiaosong Ma, and Daniel Sanchez. 2018. Exploiting Locality in Graph Analytics through Hardware-Accelerated Traversal Scheduling. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 1–14.

[30] Anurag Mukkara, Nathan Beckmann, and Daniel Sanchez. 2017. Cache-Guided Scheduling: Exploiting Caches to Maximize Locality in Graph Processing. In *Proceedings of the International Workshop on Architecture for Graph Processing*.

[31] Muhammet Mustafa Ozdal, Serif Yesil, Taemin Kim, Andrey Ayupov, John Greth, Steven Burns, and Ozcan Ozturk. 2016. Energy Efficient Architecture for Graph Analytics Accelerators. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 166–177.

[32] Sunghyun Park, Tushar Krishna, Chia-Hsin Chen, Bhavya Daya, Anantha Chandrakasan, and Li-Shiuan Peh. 2012. Approaching the Theoretical Limits of a Mesh NoC with a 16-Node Chip Prototype in 45nm SOI. In *Proceedings of the Design Automation Conference (DAC)*. 398–405.

[33] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 77–85.

[34] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. 2016. Volumetric and Multi-view CNNs for Object Classification on 3D Data. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 5648–5656.

[35] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. 2017. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*. 5105–5114.

[36] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. 2017. OctNet: Learning Deep 3D Representations at High Resolutions. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 6620–6629.

[37] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. 2018. Mining Point Cloud Local Structures by Kernel Correlation and Graph Pooling. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 4548–4557.

[38] Dongjoo Shin, Jinmook Lee, Jinsu Lee, Juhyoung Lee, and Hoi-Jun Yoo. 2018. DNPU: An Energy-Efficient Deep-Learning Processor with Heterogeneous Multi-Core Architecture. *IEEE Micro* 38, 5 (Sep./Oct. 2018), 85–93.

[39] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 29–38.

[40] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[41] Xinkai Song, Tian Zhi, Zhe Fan, Zhenxing Zhang, Xi Zeng, Wei Li, Xing Hu, Zidong Du, Qi Guo, and Yunji Chen. 2021. Cambricon-G: A Polyvalent Energy-efficient Accelerator for Dynamic Graph Neural Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (Early Access)* (2021). https://doi.org/10.1109/TCAD.2021.3052138

[42] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. 2015. Multi-view Convolutional Neural Networks for 3D Shape Recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 945–953.

[43] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 2818–2826.

[44] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. 2019. KPConv: Flexible and Deformable Convolution for Point Clouds. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 6410–6419.

[45] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. 2017. O-CNN: Octree-Based Convolutional Neural Networks for 3D Shape Analysis. *ACM Transactions on Graphics* 36, 4, Article 72 (July 2017).

[46] Yuke Wang, Boyuan Feng, Gushu Li, Shuangchen Li, Lei Deng, Yuan Xie, and Yufei Ding. 2021. GNNAdvisor: An Adaptive and Efficient Runtime System for GNN Acceleration on GPUs. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*. 515–531.

[47] Yue Wang and Justin M. Solomon. 2019. Deep Closest Point: Learning Representations for Point Cloud Registration. In *Proceedings of the International Conference on Computer Vision (ICCV)*. 3522–3531.

[48] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2019. Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics* 38, 5, Article 146 (Oct. 2019), 12 pages.

[49] Hao Wei, Jeffrey Xu Yu, Can Lu, and Xuemin Lin. 2016. Speedup Graph Processing by Graph Ordering. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1813–1828.

[50] Yi-Chung Wu, Chia-Hua Chang, Jui-Hung Hung, and Chia-Hsiang Yang. 2017. A 135-mW Fully Integrated Data Processor for Next-Generation Sequencing. *IEEE Transactions on Biomedical Circuits and Systems* 11, 6 (Dec. 2017), 1216–1225.

[51] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3D ShapeNets: A Deep Representation for Volumetric Shapes. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 1912–1920.

[52] Yuxing Xie, Tian Jiaojiao, and Xiao Xiang Zhu. 2020. Linking Points With Labels in 3D: A Review of Point Cloud Semantic Segmentation. *IEEE Geoscience and Remote Sensing Magazine* 8, 4 (Dec 2020), 38–59.

[53] Qiangeng Xu, Xudong Sun, Cho-Ying Wu, Panqu Wang, and Ulrich Neumann. 2020. Grid-GCN for Fast and Scalable Point Cloud Learning. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 5660–5669.

[54] Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. 2020. HyGCN: A GCN Accelerator with Hybrid Architecture. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*. 15–29.

[55] Pengcheng Yao, Long Zheng, Zhen Zeng, Yu Huang, Chuangyi Gui, Xiaofei Liao, Hai Jin, and Jingling Xue. 2020. A Locality-Aware Energy-Efficient Accelerator for Graph Mining Applications. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 895–907.

[56] Haoxuan You, Yifan Feng, Rongrong Ji, and Yue Gao. 2018. PVNet: A Joint Convolutional Network of Point Cloud and Multi-View for 3D Shape Recognition. In *Proceedings of the International Conference on Multimedia (MM)*. 1310–1318.

[57] Tan Yu, Jingjing Meng, and Junsong Yuan. 2018. Multi-view Harmonized Bilinear Network for 3D Object Recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 186–194.

[58] Pingpeng Yuan, Changfeng Xie, Ling Liu, and Hai Jin. 2016. PathGraph: A Path Centric Graph Processing System. *IEEE Transactions on Parallel and Distributed Systems* 27, 10 (Oct. 2016), 2998–3012.

[59] Jiaying Zhang, Xiaoli Zhao, Zheng Chen, and Zhejun Lu. 2019. A Review of Deep Learning-Based Semantic Segmentation for Point Cloud. *IEEE Access* 7 (Dec. 2019), 179118–179133.

[60] Kuangen Zhang, Ming Hao, Jing Wang, Clarence W. de Silva, and Chenglong Fu. 2019. Linked Dynamic Graph CNN: Learning on Point Cloud via Linking Hierarchical Features. arXiv:1904.10014