PointAcc
Efficient Point Cloud Accelerator

Yujun Lin, Zhekaï Zhang, Haotian Tang, Hanrui Wang, and Song Han

https://hanlab.mit.edu/projects/pointacc
video source: http://www.semantic-kitti.org/
2D Image

3D Point Cloud

video source: http://www.cvlibs.net/datasets/kitti/
Point Cloud Deep Learning are Everywhere

VR Glasses

Autonomous Driving Vehicles

AR iPhones and iPad

LiDAR Mapping Drones

LiDAR Scanner
Efficiency and Safety are Important
Efficiency and Safety are Important

Point Cloud Networks have **higher accuracy**, but **cannot run efficiently** on today’s GPUs (7× less #MACs but 1.3× slower).
Efficiency and Safety are Important

Can existing neural network accelerators solve the point cloud challenge?

NO

Point Cloud Networks have higher accuracy, but cannot run efficiently on today’s GPUs (7× less #MACs but 1.3× slower).
Point Cloud Convolution is Different from Convention

**Conventional Convolution**

Input sparsity is from ReLU

**Point Cloud Convolution**

Input sparsity is from the distribution in physical space

### Dataset Density

- **ImageNet**
  - (2D Images)
  - 100% dense input

- **SemanticKITTI**
  - (3D point clouds)
  - 0.01% sparse input

- **Empty Space behind Cars**

- **Empty Space**

Video source: [http://www.semantic-kitti.org/](http://www.semantic-kitti.org/)
Point Cloud Convolution is Different from Convention

Conventional Convolution

Input sparsity is from ReLU
Nonzeros will dilate

Point Cloud Convolution

Input sparsity is from the distribution in physical space
Nonzeros will not dilate

video source: https://github.com/facebookresearch/SparseConvNet
Point Cloud Convolution is Different from Convention

Conventional Convolution

Input sparsity is from ReLU
Nonzeros will dilate

Each nonzero input is multiplied with all nonzero weights

Point Cloud Convolution

Input sparsity is from the distribution in physical space
Nonzeros will not dilate

Each nonzero input is not multiplied with all nonzero weights
Point Cloud Convolution is Different from Convention

- **Sparsity in Physical Space**: Input sparsity is from the point distribution in physical space.
- **Nonzeros will not dilute**: Nonzeros will not dilute.
- **Irregular and sparse computation pattern**: Each nonzero input are not multiplied with all nonzero weights.
- **New operations to find neighbors**: Mapping operations.
- **Explicit data movement overhead**: Gather features and Scatter psums.

### Mapping Operations
- **Output Cloud Construction**
- **Neighbor Search**
- **Feature Gather**
- **Feature Transformation**
- **Scatter Reduction**

### MatMul Operations
- **Sparsity in Physical Space**
- **Nonzeros will not dilute**
- **Irregular and sparse computation pattern**
- **New operations to find neighbors**
- **Explicit data movement overhead**
Introduction to Point Cloud Convolution

Output Cloud Construction

Neighbor Search

Feature Gather

Feature Transformation

Scatter Reduction

MatMul Operations
Step 1: Build Output Point Cloud

1. **Output Cloud Construction**
2. **Neighbor Search**
3. **Feature Gather**
4. **Feature Transformation**
5. **Scatter Reduction**

- **Mapping Operations**
- **MatMul Operations**

Input Point Cloud:

- \((P_0, F_0)\)
- \((P_1, F_1)\)
- \((P_2, F_2)\)
- \((P_3, F_3)\)
- \((P_4, F_4)\)

(Coords, Feature Vector)

Output Point Cloud:

- \(Q_0\)
- \(Q_1\)
- \(Q_2\)
- \(Q_3\)
- \(Q_4\)

Downsample process:

- \(P_0\) to \(Q_0\)
- \(P_1\) to \(Q_1\)
- \(P_2\) to \(Q_2\)
- \(P_3\) to \(Q_3\)
- \(P_4\) to \(Q_4\)

Upsample process:

- \(Q_0\) to \(P_0\)
- \(Q_1\) to \(P_1\)
- \(Q_2\) to \(P_2\)
- \(Q_3\) to \(P_3\)
- \(Q_4\) to \(P_4\)
Step 2: Search Neighbors

For each output point, we find the input points in its neighborhood and generate (Input Point, Output Point, Weight Index) maps.

**Input Point Cloud**

\((P_0, F_0)\)  
\((P_1, F_1)\)  
\((P_2, F_2)\)  
\((P_3, F_3)\)  
\((P_4, F_4)\)

(Coords, Feature Vector)

**Output Point Cloud**

\(Q_0\)  
\(Q_1\)  
\(Q_2\)  
\(Q_3\)  
\(Q_4\)

**Neighborhood Shape**

<table>
<thead>
<tr>
<th>Ball</th>
<th>Cube</th>
<th>Nearest Neighbor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ball Query</td>
<td>Kernel Mapping</td>
<td>k-Nearest-Neighbors</td>
</tr>
</tbody>
</table>

**Operational Flow**

- Output Cloud Construction
- Neighbor Search
- Feature Gather
- Feature Transformation
- Scatter Reduction

**Mapping Operations**

MatMul Operations
Step 3: Gather Input Features

Input Point Cloud

(Q0, F0)
(Q1, F1)
(Q2, F2)
(Q3, F3)
(Q4, F4)

Maps (In, Out, Wgt)

(P0, Q1, W1,-1)
(P3, Q4, W1,-1)
(P1, Q3, W1,0)
(P0, Q0, W0,0)
(P1, Q1, W0,0)
(P2, Q2, W0,0)
(P3, Q3, W0,0)
(P4, Q4, W0,0)
(P1, Q0, W1,0)
(P3, Q1, W1,0)
(P1, Q0, W1,1)
(P4, Q3, W1,1)

Gather By Weight

F0

F3

W1,-1

F1

F1

W1,0

F0

W0,0

F2

F3

W1,0

F3

W1,1

F4

Output Point Cloud

Q0
Q1
Q2
Q3
Q4
Step 4: Transform Input Features

**Output Cloud Construction**
- Mapping Operations

**Neighbor Search**
- Feature Gather
- Feature Transformation
- Scatter Reduction

**Feature Gather**
- MatMul Operations

### Mapping Operations

<table>
<thead>
<tr>
<th>Maps (In, Out, Wgt)</th>
<th>Gather By Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P₀, Q₁, W₁,⁻¹)</td>
<td>F₀ × W₁,⁻¹ = Psum₁</td>
</tr>
<tr>
<td>(P₃, Q₄, W₁,⁻¹)</td>
<td>F₃ × W₁,⁻¹ = Psum₄</td>
</tr>
<tr>
<td>(P₁, Q₃, W₁,₀)</td>
<td>F₁ × W₁,₀ = Psum₃</td>
</tr>
<tr>
<td>(P₀, Q₀, W₀,₀)</td>
<td>F₀ × W₀,₀ = Psum₀</td>
</tr>
<tr>
<td>(P₁, Q₁, W₀,₀)</td>
<td>F₁ × W₀,₀ = Psum₁</td>
</tr>
<tr>
<td>(P₂, Q₂, W₀,₀)</td>
<td>F₂ × W₀,₀ = Psum₂</td>
</tr>
<tr>
<td>(P₃, Q₃, W₀,₀)</td>
<td>F₃ × W₀,₀ = Psum₃</td>
</tr>
<tr>
<td>(P₄, Q₄, W₀,₀)</td>
<td>F₄ × W₀,₀ = Psum₄</td>
</tr>
<tr>
<td>(P₁, Q₀, W₁,₁)</td>
<td>F₀ × W₁,₁ = Psum₀</td>
</tr>
<tr>
<td>(P₄, Q₃, W₁,₁)</td>
<td>F₄ × W₁,₁ = Psum₃</td>
</tr>
</tbody>
</table>

**Matrix-Matrix Multiplication**
- \( p \)-by-ic \( \times \) ic-by-oc \( \times \) \( p \)-by-oc

**Input Point Cloud**
- (Coords, Feature Vector)

**Output Point Cloud**
- Q₀, Q₁, Q₂, Q₃, Q₄
Step 5: Scatter and Reduce Partial Sums

Output Cloud Construction
Neighbor Search
Feature Gather
Feature Transformation
Scatter Reduction

Mapping Operations
MatMul Operations

Maps (In, Out, Wgt)

<table>
<thead>
<tr>
<th>Maps</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P₀, Q₁, W₁,-1)</td>
</tr>
<tr>
<td>(P₃, Q₄, W₁,-1)</td>
</tr>
<tr>
<td>(P₁, Q₃, W₁,0)</td>
</tr>
<tr>
<td>(P₀, Q₀, W₀,0)</td>
</tr>
<tr>
<td>(P₁, Q₁, W₀,0)</td>
</tr>
<tr>
<td>(P₂, Q₂, W₀,0)</td>
</tr>
<tr>
<td>(P₃, Q₃, W₀,0)</td>
</tr>
<tr>
<td>(P₄, Q₄, W₀,0)</td>
</tr>
<tr>
<td>(P₁, Q₀, W₁,1)</td>
</tr>
<tr>
<td>(P₃, Q₁, W₁,0)</td>
</tr>
</tbody>
</table>

Gather By Weight

F₀ × W₁,-1 = Psum₁
F₀ × W₁,0 = Psum₃
F₁ × W₀,0 = Psum₀
F₂ × W₀,0 = Psum₁
F₃ × W₀,0 = Psum₂
F₄ × W₀,0 = Psum₃
F₁ × W₁,0 = Psum₁
F₄ × W₁,1 = Psum₀
F₄ × W₁,1 = Psum₃

Reduction

MaxPool, Accumulation, …

Input Point Cloud
(Q₀, F₀)
(Q₁, F₁)
(Q₂, F₂)
(Q₃, F₃)
(Q₄, F₄)

Output Point Cloud

(Coords, Feature Vector)
Point Cloud Deep Learning is Different

<table>
<thead>
<tr>
<th>Source of Sparsity</th>
<th>Point Cloud NN</th>
<th>CNN</th>
<th>Graph CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Sparsity</td>
<td>the distribution of points in physical space</td>
<td>ReLU and weight pruning</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td><strong>nonzeros do not dilate</strong>; input sparsity <strong>keeps low</strong></td>
<td>nonzeros dilate; input sparsity quickly reduces</td>
<td>-</td>
</tr>
</tbody>
</table>

- Extreme low utilization on existing sparse CNN accelerators
### Point Cloud Deep Learning is Different

<table>
<thead>
<tr>
<th>Source of Sparsity</th>
<th>CNN</th>
<th>Graph CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Sparsity</td>
<td>nonzeros do not dilate; input sparsity keeps low</td>
<td>nonzeros dilate; input sparsity quickly reduces</td>
</tr>
<tr>
<td>Receptive Field</td>
<td>both outputs’ coords and neighbors need to be explicitly calculated</td>
<td>outputs’ and neighbors’ coords are inferred by pointer arithmetic</td>
</tr>
</tbody>
</table>

- Existing NN accelerators do not support mapping ops
## Point Cloud Deep Learning is Different

<table>
<thead>
<tr>
<th>Point Cloud NN</th>
<th>CNN</th>
<th>Graph CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>source of sparsity</strong></td>
<td>the distribution of points in physical space</td>
<td>ReLU and weight pruning</td>
</tr>
<tr>
<td><strong>input sparsity</strong></td>
<td>nonzeros do not dilate; input sparsity keeps low</td>
<td>nonzeros dilate; input sparsity quickly reduces</td>
</tr>
<tr>
<td><strong>receptive field</strong></td>
<td>both outputs’ coords and neighbors need to be explicitly calculated</td>
<td>outputs’ and neighbors’ coords are inferred by pointer arithmetic</td>
</tr>
<tr>
<td><strong>weights</strong></td>
<td>weights can be shared or different for neighbors</td>
<td>weight are different for neighbors</td>
</tr>
<tr>
<td><strong>data movement</strong></td>
<td>require both gather and scatter</td>
<td>no explicit gather or scatter</td>
</tr>
</tbody>
</table>

- GCN accelerators and previous PointNet accelerator Mesorasi do not support
Challenge of Point Cloud Deep Learning

- Due to extreme sparsity, mapping operations and data movement overhead together take up >50% of the total runtime latency.

- Due to unsupported mapping ops, data movement between co-processors (CPU and TPU) worsens the bottleneck.
PointAcc: Efficient Point Cloud Accelerator

- Mapping Unit
  - Coordinates
  - Output Cloud Construction
  - Neighbor Search

- Memory Management Unit (MMU)
  - Feature Gather
  - Address

- Matrix Unit
  - Feature Transformation
  - Scatter Reduction

- Global Buffers
  - Maps
  - Features & Weights

Mapping Operations

MatMul Operations
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

Goal of Mapping Ops:
Generate Map (input point, output point, weight index)

Key Observation:
Maps are constructed based on the comparison among distances
**Mapping Unit: Diverse Mapping Ops in One Versatile Arch**

**Goal of Mapping Ops:** Generate Map (input point, output point, weight index)

**Key Observation:** Maps are constructed based on the **comparison** among distances

**Kernel Mapping**

- comparison op: not greater than, $|\Delta x| \leq 1$, $|\Delta y| \leq 1$, $|\Delta z| \leq 1$
- query the hash table of input point cloud for each output point

\[
\text{For } Q \text{ in } O = \{Q_0, Q_1, Q_2, \ldots\}:
\quad \text{Find } P \text{ in } I, \text{ s.t. } |P_x - Q_x| \leq 1 \text{ and } |P_y - Q_y| \leq 1 \text{ and } |P_z - Q_z| \leq 1
\]

\[
\text{For } W \text{ in } \{W_{-1,-1}, W_{-1,0}, W_{1,0}, \ldots\}: \\
\quad \text{For } Q \text{ in } O = \{Q_0, Q_1, Q_2, \ldots\}: \\
\quad \quad \text{Query } Q + \delta_W \text{ in HashTable}(I)
\]

- require on-chip memory as large as 160MB
- cannot be parallelized efficiently
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

Goal of Mapping Ops: Generate Map (input point, output point, weight index)

Key Observation: Maps are constructed based on the comparison among distances

Kernel Mapping

- comparison op: not greater than, $|\Delta x| \leq 1$, $|\Delta y| \leq 1$, $|\Delta z| \leq 1$
- query the hash table of input point cloud for each output point
- coordinates intersection for each neighbor position
- parallelizable merge-sort and equal comparison

Input Point Cloud

Output Point Cloud

Shift Input for $W_{-1,-1}$

Merge Sort

Intersection
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

Goal of Mapping Ops: Generate Map (input point, output point, weight index)

Key Observation: Maps are constructed based on the comparison among distances

Kernel Mapping

- comparison op: not greater than, $|\Delta x| \leq 1$, $|\Delta y| \leq 1$, $|\Delta z| \leq 1$
- query the hash table of input point cloud for each output point
- coordinates intersection for each neighbor position
- $\rightarrow$ parallezable merge-sort and equal comparison
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

Goal of Mapping Ops: Generate Map (input point, output point, weight index)

Key Observation: Maps are constructed based on the comparison among distances

Kernel Mapping

- comparison op: not greater than, $|\Delta x| \leq 1$, $|\Delta y| \leq 1$, $|\Delta z| \leq 1$
- query the hash table of input point cloud for each output point
- coordinates intersection for each neighbor position
- $\rightarrow$ parallezable merge-sort and equal comparison

### Input Point Cloud

<table>
<thead>
<tr>
<th>$P_0$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_0$</td>
<td>$Q_1$</td>
<td>$Q_2$</td>
<td>$Q_3$</td>
<td>$Q_4$</td>
</tr>
</tbody>
</table>

| stride = 1 |

### Output Point Cloud

<table>
<thead>
<tr>
<th>$W_{1,-1}$</th>
<th>$W_{1,0}$</th>
<th>$W_{1,1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_{0,-1}$</td>
<td>$W_{0,0}$</td>
<td>$W_{0,1}$</td>
</tr>
<tr>
<td>$W_{1,-1}$</td>
<td>$W_{1,0}$</td>
<td>$W_{1,1}$</td>
</tr>
</tbody>
</table>

### Merge Sort

<table>
<thead>
<tr>
<th>$P_0$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_0$</td>
<td>$Q_1$</td>
<td>$Q_2$</td>
<td>$Q_3$</td>
<td>$Q_4$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$P_0$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_0$</td>
<td>$Q_1$</td>
<td>$Q_2$</td>
<td>$Q_3$</td>
<td>$Q_4$</td>
</tr>
</tbody>
</table>

### Shift Input for $W_{1,1}$

**Example Shifts:**

- $(P_0, Q_1, W_{1,-1})$
- $(P_3, Q_4, W_{1,-1})$
- $(P_1, Q_0, W_{1,1})$
- $(P_4, Q_3, W_{1,1})$
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

**Goal of Mapping Ops:** Generate Map (input point, output point, weight index)

**Key Observation:** Maps are constructed based on the *comparison* among distances

**Kernel Mapping**
- comparison op: not greater than, $|\Delta x| \leq 1$, $|\Delta y| \leq 1$, $|\Delta z| \leq 1$
- query the hash table of input point cloud for each output point
- coordinates intersection for each neighbor position
- → parallezable merge-sort and equal comparison
Goal of Mapping Ops: Generate Map (input point, output point, weight index)

Key Observation: Maps are constructed based on the comparison among distances

**k-Nearest-Neighbor / Ball Query**
- comparison op: TopK
- ball query filters out the outsider in the nearest neighbors

**Farthest Point Sampling**
- comparison op: ArgMax / Max
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

**Goal of Mapping Ops:** Generate Map (input point, output point, weight index)

**Key Observation:** Maps are constructed based on the *comparison* among distances

---

**Farthest Point Sampling**

1. Input Point Cloud
2. Distances to Output Point Cloud
3. Input Point Cloud
   - DIST
   - MIN
   - Output Point Cloud

**K Nearest Neighbor / Ball Query**

1. Input Point Cloud
2. TopK
   - DIST
   - Output Point Cloud
   - Neighbors (Maps)

**Kernel Mapping**

1. Input Point Cloud
2. QUANTIZE
   - Shifted Input Point Cloud
3. MergeSort
   - Neighbors (Maps)
   - Output Point Cloud
**Mapping Unit: Diverse Mapping Ops in One Versatile Arch**

- **Sorter Buffer**
- **Distance Calculation**
  - \( \text{arg max} \)
  - \( \text{min} \)
  - Update Distance
- **N/2 Sorter**
- **Merger Buffer**
- **N Merger**
- **MergeSort (MS)**
- **Detect Intersection (DI)**

**Operations:**
- **Load Input Points**
- **Iterative MergeSort Distances**
- **Get Next Output Point**
- **Calculate Distance (CD)**
- **Sort (ST)**
- **Fetch Coords (FS)**
- **Fetch Coords (FM)**
- **DRAM**
- **Store Maps**

**Kernels:**
- Kernel Mapping
- k Nearest Neighbor
- Farthest Point Sampling
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

- Main component for parallel comparison: sorters, merger
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

- Data flow when running kernel mapping (i.e., the neighborhood shape is cube)
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

- Data flow when running k-nearest-neighbor / ball query (i.e., the neighborhood shape is ball)
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

- Data flow when running farthest point sampling
Flexible Memory Management

Sparse Computation

Dense Computation

Streaming Computation with Caching

Temporal Layer Fusion
Sparse MatMul Operations

the state-of-the-art GPU implementation
Streaming Sparse MatMul Operations

the state-of-the-art GPU implementation

PointAcc implementation
Streaming Sparse MatMul Operations

- Sequential fetch input features on demand in the granularity of tile (i.e., “cache block”)
  - No more random access for gathering features
Streaming Sparse MatMul Operations

- Sequential fetch input features on demand in the granularity of tile (i.e., “cache block”)
  - No more random access for gathering features
- MatMul computing parallelizes input channel (\(ic\)) and output channel (\(oc\)) dimension

PointAcc implementation
Streaming Sparse MatMul Operations

- Sequential fetch input features on demand in the granularity of tile \((i.e., \text{“cache block”})\)
  - No more random access for gathering features
- MatMul computing parallelizes input channel \((ic)\) and output channel \((oc)\) dimension
  - No need for on-chip scatter network for scattering psums of different points simultaneously
Streaming Sparse MatMul Operations

- Sequential fetch input features on demand in the granularity of tile (i.e., “cache block”)
- MatMul computing parallelizes $ic$ and $oc$ dimension → no need for on-chip scatter network

PointAcc implementation
Streaming Sparse MatMul Operations

- Sequential fetch input features on demand in the granularity of tile (i.e., “cache block”)
- MatMul computing parallelizes $ic$ and $oc$ dimension $\rightarrow$ no need for on-chip scatter network
- Weight stationary saves the on-chip memory footprint
  - Key: #points ($10^3 \sim 10^5$) $\gg$ #channels ($10 \sim 10^3$)
Streaming Sparse MatMul Operations

- Sequential fetch input features on demand in the granularity of tile (i.e., “cache block”)
- MatMul computing parallelizes $ic$ and $oc$ dimension $\rightarrow$ no need for on-chip scatter network
- Weight stationary saves the on-chip memory footprint

PointAcc implementation
Streaming Sparse MatMul Operations

- Sequential fetch input features on demand in the granularity of tile (i.e., “cache block”)
- MatMul computing parallelizes $ic$ and $oc$ dimension → no need for on-chip scatter network
- Weight stationary saves the on-chip memory footprint
- Caching reduces the #off-chip read of reused data to nearly 1 access
Streaming Sparse MatMul Operations

- Sequential fetch input features on demand in the granularity of tile (i.e., “cache block”)
- MatMul computing parallelizes \( ic \) and \( oc \) dimension → no need for on-chip scatter network
- Weight stationary saves the on-chip memory footprint
- Caching reduces the #off-chip read of reused data to nearly 1 access
Streaming Sparse MatMul Operations

- Sequential fetch input features on demand in the granularity of tile (i.e., “cache block”)
- MatMul computing parallelizes $ic$ and $oc$ dimension → no need for on-chip scatter network
- Weight stationary saves the on-chip memory footprint
- Caching reduces the #off-chip read of reused data to nearly 1 access
Streaming Sparse MatMul Operations

- Sequential fetch input features on demand in the granularity of tile (i.e., “cache block”)
- MatMul computing parallelizes $ic$ and $oc$ dimension $\rightarrow$ no need for on-chip scatter network
- Weight stationary saves the on-chip memory footprint
- Caching reduces the #off-chip read of reused data to nearly 1 access
- Output stationary eliminates the off-chip scattering of partial sums
Streaming Sparse MatMul Operations

the state-of-the-art GPU implementation

PointAcc implementation
Flexible Memory Management

Dense Computation

```
Layer 0
p0 p1 p2 p3 p4 p5 p6 p7
FC
Layer 1
p0 p1 p2 p3 p4 p5 p6 p7
FC
Layer 2
p0 p1 p2 p3 p4 p5 p6 p7
FC
Layer 3
p0 p1 p2 p3 p4 p5 p6 p7
FC
```

Temporal Layer Fusion

Streaming Computation with Caching

Sparse Computation

Spatial Layer Fusion

Temporal Layer Fusion
Step 4: Transform Input Features

Output Cloud Construction
Neighbor Search
Feature Gather
Feature Transformation
Scatter Reduction

Mapping Operations
MatMul Operations

Input Point Cloud
Output Point Cloud

Maps (In, Out, Wgt)

(Coords, Feature Vector)

\[(P_0, F_0), (P_1, F_1), (P_2, F_2), (P_3, F_3), (P_4, F_4)\]

\[
\begin{align*}
(P_0, Q_1, W_{-1,-1}) \\
(P_3, Q_4, W_{-1,-1}) \\
(P_1, Q_3, W_{-1,0}) \\
(P_0, Q_0, W_{0,0}) \\
(P_1, Q_1, W_{0,0}) \\
(P_2, Q_2, W_{0,0}) \\
(P_3, Q_3, W_{0,0}) \\
(P_4, Q_4, W_{0,0}) \\
(P_3, Q_1, W_{1,0}) \\
(P_1, Q_0, W_{1,1}) \\
(P_4, Q_3, W_{1,1})
\end{align*}
\]

Gather By Weight

\[\begin{align*}
F_0 \times W_{-1,-1} & = P_{sum1} \\
F_3 \times W_{-1,0} & = P_{sum3} \\
F_1 \times W_{0,0} & = P_{sum0} \\
F_4 \times W_{0,1} & = P_{sum2} \\
F_3 \times W_{1,0} & = P_{sum1} \\
F_4 \times W_{1,1} & = P_{sum0}
\end{align*}\]

Matrix-Matrix Multiplication

\[p-by-ic \quad ic-by-oc \quad p-by-oc\]
Layer Fusion

Spatial Layer Fusion

- Core 1 → Buffer → Core 2 → Buffer → Core 3

- Core 1: Layer 1, Layer 1
- Core 2: Layer 2, Layer 2
- Core 3: Layer 3, Layer 3

- ✓ Reduced off-chip memory access
- ✗ Complicated logic overhead
- ✗ Inflexible intermediate buffer use
- ✗ Fixed #layers for fusion

Temporal Layer Fusion

- Core → Buffer

- Large Core: Layer 1, Layer 2, Layer 3, Layer 4, Layer 3, Layer 4

- ✓ Reduced off-chip memory access
- ✓ Simple logic modification
- ✓ Flexible buffer use for better layer tiling
- ✓ Flexible #layers for fusion

# consecutive FCs varies among different point cloud NNs, and even among different blocks of the same model
Temporal Layer Fusion on consecutive FCs

DRAM access per point
- read
- write

Tile 0 (Layer 0 Input)

Input Buffers (Stack mode)
- \( p_0 \)
- \( p_1 \)
- \( \ldots \)
- \( p_{31} \)
- \( p_{32} \)
- \( p_{33} \)
- \( \ldots \)
- \( p_{63} \)

Output Buffers
- \( p_0 \)
- \( p_1 \)
- \( \ldots \)
- \( p_{31} \)
- \( p_{32} \)
- \( p_{33} \)
- \( \ldots \)
- \( p_{63} \)

#channels = 64

Without layer fusion

With layer fusion

FC

Layer 0

#channels = 64

DRAM access per point
- read
- write

#channels = 64
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)

Output Buffers

Tile 0 (Layer 0 Input)

#channels = 64

Layer 0

#channels = 64

DRAM access per point

read

write

without layer fusion

with layer fusion
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)

Output Buffers

Tile 0 (Layer 0 Input)

#channels = 64

Layer 0

#channels = 64

Layer 1

#channels = 128

DRAM access per point

without layer fusion

with layer fusion

without

read

write

with layer fusion
Temporal Layer Fusion on consecutive FCs

**Input Buffers (Stack mode)**

- \( P_0 \)
- \( P_1 \)
- \( \ldots \)
- \( P_{31} \)
- \( P_{32} \)
- \( P_{33} \)
- \( \ldots \)
- \( P_{63} \)

**Output Buffers**

- \( P_0 \)
- \( P_1 \)
- \( \ldots \)
- \( P_{31} \)
- \( P_{32} \)
- \( P_{33} \)
- \( \ldots \)
- \( P_{63} \)

**DRAM access per point**

- without layer fusion
- with layer fusion

**Tile 0**

(Layer 0 Input)

\( p_0 \) \( p_1 \) \( \ldots \) \( p_{31} \) \( p_{32} \) \( p_{33} \) \( \ldots \) \( p_{63} \)

- \( \#\text{channels} = 64 \)

**Layer 0**

\( p_0 \) \( p_1 \) \( \ldots \) \( p_{31} \) \( p_{32} \) \( p_{33} \) \( \ldots \) \( p_{63} \) \( p_{64} \) \( p_{65} \) \( \ldots \) \( p_{512} \)

- \( \#\text{channels} = 64 \)

**Layer 1**

\( p_0 \) \( p_1 \) \( \ldots \) \( p_{31} \) \( p_{32} \) \( p_{33} \) \( \ldots \) \( p_{63} \) \( p_{64} \) \( p_{65} \) \( \ldots \) \( p_{512} \)

- \( \#\text{channels} = 128 \)
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)

Output Buffers

#channels = 64

Layer 0

Layer 1

#channels = 64

#channels = 128

without layer fusion

with layer fusion

DRAM access per point

read

write
Temporal Layer Fusion on consecutive FCs

**Input Buffers (Stack mode)**

```
P0
P1
...
P31
P32
P33
...
P63
```

**Output Buffers**

```
```

**Tile 1**

(Layer 1 Input)

**DRAM access per point**

- **read**
- **write**

**without**

layer fusion

**with**

layer fusion

**#channels = 64**

**FC**

```
p0  p1  …  p31  p32  p33  …  p63  p64  p65  …  p512
```

**Layer 0**

**#channels = 64**

**FC**

```
p0  p1  …  p31  p32  p33  …  p63  p64  p65  …  p512
```

**Layer 1**

**#channels = 128**
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)

Output Buffers

Tile 1 (Layer 1 Input)

#channels = 64

#channels = 64

#channels = 128

DRAM access per point
- read
- write

without layer fusion

with layer fusion

without Layer 1 Input

with Layer 1 Input

Layer 0

Layer 1
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)

Output Buffers

Tile 1 (Layer 1 Input)

DRAM access per point

without layer fusion

with layer fusion

#channels = 64

#channels = 64

#channels = 128

#channels = 128
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)

Output Buffers

Tile 1 (Layer 1 Input)

DRAM access per point

without layer fusion

with layer fusion

#channels = 64

#channels = 64

#channels = 128

#channels = 128
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)

Output Buffers

Tile 1 (Layer 1 Input)

#channels = 64

Layer 0

#channels = 64

Layer 1

#channels = 128

Layer 2

#channels = 128
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)

Output Buffers

Tile 1
(Layer 1 Input)

Tile 2
(Layer 2 Input)

DRAM access per point

without layer fusion

with layer fusion

#channels = 64

#channels = 64

#channels = 128

#channels = 128
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)

Output Buffers

Tile 1 (Layer 1 Input)

Tile 2 (Layer 2 Input)

DRAM access per point

Without layer fusion

With layer fusion

#channels = 64

#channels = 64

#channels = 128

#channels = 128

$p_0 \ p_1 \ \ldots \ p_{31} \ p_{32} \ p_{33} \ \ldots \ p_{63} \ | \ p_{64} \ p_{65} \ \ldots \ p_{512}$

$p_0 \ p_1 \ \ldots \ p_{31} \ | \ p_{32} \ p_{33} \ \ldots \ p_{63} \ | \ p_{64} \ p_{65} \ \ldots \ p_{512}$

$p_0 \ p_1 \ \ldots \ p_{31} \ | \ p_{32} \ p_{33} \ \ldots \ p_{63} \ | \ p_{64} \ p_{65} \ \ldots \ p_{512}$

$p_0 \ p_1 \ \ldots \ p_{31} \ | \ p_{32} \ p_{33} \ \ldots \ p_{63} \ | \ p_{64} \ p_{65} \ \ldots \ p_{512}$
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)

Output Buffers

Tile 2
(Layer 2 Input)

Input Buffers

Output Buffers

Tile 1
(Layer 1 Input)

Tile 2
(Layer 2 Input)

Tile 1
(Layer 1 Input)

Layer 0

Layer 1

Layer 2

#channels = 64

#channels = 64

#channels = 128

#channels = 128

without layer fusion

with layer fusion

DRAM access per point

read

write

#channels = 64

#channels = 64
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)

Output Buffers

Tile 1 (Layer 1 Input)

Tile 2 (Layer 2 Input)

DRAM access per point

without layer fusion

with layer fusion

#channels = 64

Layer 0

Layer 1

Layer 2

#channels = 128

#channels = 128

#channels = 64

Without

With

read

write
Temporal Layer Fusion on consecutive FCs

Drum access per point

without layer fusion  with layer fusion

Tile 1 (Layer 1 Input)

Input Buffers (Stack mode)

Output Buffers

#channels = 64

Layer 0

Layer 1

Layer 2

#channels = 64

#channels = 128

#channels = 128
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)

Output Buffers

Tile 1 (Layer 1 Input)

#channels = 64

Layer 0

FC

Layer 1

FC

Layer 2

FC

#channels = 128

#channels = 128

#channels = 128

#channels = 64

#channels = 64

#channels = 64

#channels = 64

without layer fusion

with layer fusion

DRAM access per point

read

write
**Temporal Layer Fusion on consecutive FCs**

<table>
<thead>
<tr>
<th>Layer 0</th>
<th>Layer 1</th>
<th>Layer 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_0 ) ( p_1 ) ( \ldots ) ( p_{31} ) ( p_{32} ) ( p_{33} ) ( \ldots ) ( p_{63} ) ( p_{64} ) ( p_{65} ) ( \ldots ) ( p_{512} )</td>
<td>( p_0 ) ( p_1 ) ( \ldots ) ( p_{31} ) ( p_{32} ) ( p_{33} ) ( \ldots ) ( p_{63} ) ( p_{64} ) ( p_{65} ) ( \ldots ) ( p_{512} )</td>
<td>( p_0 ) ( p_1 ) ( \ldots ) ( p_{31} ) ( p_{32} ) ( p_{33} ) ( \ldots ) ( p_{63} ) ( p_{64} ) ( p_{65} ) ( \ldots ) ( p_{512} )</td>
</tr>
</tbody>
</table>

**Input Buffers** (Stack mode)

- \( p_{32} \)
- \( p_{33} \)
- \( \ldots \)
- \( p_{63} \)

**Output Buffers**

- \( P_{32} \)
- \( P_{33} \)
- \( \ldots \)
- \( P_{63} \)

**Tile 1** (Layer 1 Input)

- \( P_0 \)
- \( P_1 \)
- \( \ldots \)
- \( P_{63} \)

**DRAM access per point**

- read
- write

**#channels = 64**

**#channels = 128**
Temporal Layer Fusion on consecutive FCs

DRAM access per point
- read
- write

Tile 1 (Layer 1 Input)

Input Buffers (Stack mode)

Output Buffers

#channels = 64

Layer 0

Layer 1

Layer 2

#channels = 64

#channels = 128

#channels = 128
Temporal Layer Fusion on consecutive FCs

**Input Buffers (Stack mode)**

- P_{32}
- P_{33}
- ...
- P_{63}

**Output Buffers**

- P_{32}
- P_{33}
- ...
- P_{63}

**Tile 1 (Layer 1 Input)**

Without layer fusion:

- DRAM access per point
  - Read
  - Write

With layer fusion:

- DRAM access per point
  - Read
  - Write

#channels = 64

Layer 0

Layer 1

Layer 2

#channels = 64

#channels = 128

#channels = 128
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)  Output Buffers

Layer 0

#channels = 64

Layer 1

#channels = 64

Layer 2

#channels = 128

#channels = 128

DRAM access per point

without layer fusion  with layer fusion

read  write

without layer fusion  with layer fusion

#channels = 64
Temporal Layer Fusion on consecutive FCs

DRAM access per point
- read
- write

Tile 2 (Layer 2 Input)

Input Buffers (Stack mode)

Output Buffers

#channels = 64

Layer 0

Layer 1

Layer 2

#channels = 128

without layer fusion

with layer fusion

#channels = 64

#channels = 128

#channels = 128
Temporal Layer Fusion on consecutive FCs

DRAM access per point

without layer fusion  with layer fusion

Tile 2 (Layer 2 Input)

Input Buffers (Stack mode)

Output Buffers

#channels = 64

#channels = 64

#channels = 128

#channels = 128
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)

Output Buffers

DRAM access per point

without layer fusion

with layer fusion

Tiles:

1. Layer 0
   - #channels = 64
   - Layer 0
   - FC
   - \( p_0, p_1, \ldots, p_{31}, p_{32}, p_{33}, \ldots, p_{63} \)

2. Layer 1
   - #channels = 64
   - Layer 1
   - FC
   - \( p_0, p_1, \ldots, p_{31}, p_{32}, p_{33}, \ldots, p_{63} \)

3. Layer 2
   - #channels = 128
   - Layer 2
   - FC
   - \( p_0, p_1, \ldots, p_{31}, p_{32}, p_{33}, \ldots, p_{63} \)

4. Layer 3
   - #channels = 128
   - Layer 3
   - FC
   - \( p_0, p_1, \ldots, p_{31}, p_{32}, p_{33}, \ldots, p_{63} \)

#channels = 64

#channels = 64

#channels = 128

#channels = 128
### Temporal Layer Fusion on consecutive FCs

**Input Buffers (Stack mode)**

- $P_{32}$
- $P_{32}$
- $P_{33}$
- $P_{33}$
- ...
- ...
- $P_{63}$
- $P_{63}$

**Output Buffers**

- $P_{32}$
- $P_{32}$
- $P_{33}$
- $P_{33}$
- ...
- ...
- $P_{63}$
- $P_{63}$

- **#channels = 64**
- **Layer 0**
- **Layer 1**
- **Layer 2**

- **#channels = 64**
- **#channels = 128**
- **#channels = 128**

**Tile 2**

(Layer 2 Input)

**DRAM access per point**

- without layer fusion
- with layer fusion

- read
- write

- 3x
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)

Output Buffers

P32
P32
P32
...
P63
P63

#channels = 64

#channels = 64

#channels = 128

#channels = 128

DRAM access per point

without layer fusion

with layer fusion

read

write

3×
Temporal Layer Fusion on consecutive FCs

Input Buffers (Stack mode)

Output Buffers

#channels = 64

Without layer fusion

Read

Write

3×

With layer fusion

Layer 0

Layer 1

Layer 2

#channels = 64

#channels = 128

#channels = 128
PointAcc: Efficient Point Cloud Accelerator

Global Buffers

Mapping Unit
- Fetch Coords
- Calculate Distance
- Split & Sort
- Buffering
- MergeSort
- Detect Intersection

Maps
- Coordinates
- Features & Weights

Memory Management Unit (MMU)
- Map FIFO
- Memory Meta Container
- Loop Counters
- Addr. Gen

Systolic Array

Matrix Unit

Mapping Operations

MatMul Operations
### Evaluation Setup

#### Evaluation benchmarks

- 4 different applications
- 5 different datasets
- 8 different point cloud models

<table>
<thead>
<tr>
<th>Application</th>
<th>Dataset</th>
<th>Scene</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification</td>
<td>ModelNet40</td>
<td>Object</td>
<td>PointNet</td>
</tr>
<tr>
<td>Part Segmentation</td>
<td>ShapeNet</td>
<td></td>
<td>PointNet++ (c)</td>
</tr>
<tr>
<td>Detection</td>
<td>KITTI</td>
<td>Outdoor</td>
<td>PointNet++ (ps)</td>
</tr>
<tr>
<td>Semantic Segmentation</td>
<td>S3DIS</td>
<td>Indoor</td>
<td>DGCNN</td>
</tr>
<tr>
<td></td>
<td>SemanticKITT</td>
<td>Outdoor</td>
<td>MinkNet(o)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F-PointNet++</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PointNet++(s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MinkNet(i)</td>
</tr>
</tbody>
</table>
Evaluation Setup

Evaluation benchmarks

- 4 different applications: classification, part segmentation, detection, semantic segmentation
- 5 different datasets, ranging from single object to indoor scenes to outdoor scenes
- 8 different point cloud models, including classical and the state-of-the-art ones

Hardware Baselines

- server-level products: Intel Xeon CPU, RTX 2080Ti GPU, TPU v3
- edge devices: Jetson Xavier NX, Jetson Nano, Raspberry Pi
- specialized point cloud NN ASIC: Mesorasi

Variants

- PointAcc: 64 × 64 systolic array with 776 KB on-chip memory
- PointAcc.Edge: 16 × 16 systolic array with 274 KB on-chip memory
# Performance Gain over the Server Products

<table>
<thead>
<tr>
<th>Model</th>
<th>NVIDIA RTX 2080Ti</th>
<th>Intel Xeon Skylake + TPU V3</th>
<th>Intel Xeon Gold 6130</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet</td>
<td>3.7</td>
<td>2.8</td>
<td>3.7</td>
</tr>
<tr>
<td>PointNet++ (c)</td>
<td>127</td>
<td>113</td>
<td>97</td>
</tr>
<tr>
<td>PointNet++ (ps)</td>
<td>113</td>
<td>97</td>
<td>127</td>
</tr>
<tr>
<td>DGCNN</td>
<td>82</td>
<td>82</td>
<td>82</td>
</tr>
<tr>
<td>F-PointNet++</td>
<td>65</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>PointNet++ (s)</td>
<td>269</td>
<td>269</td>
<td>269</td>
</tr>
<tr>
<td>MinkNet (i)</td>
<td>131</td>
<td>131</td>
<td>131</td>
</tr>
<tr>
<td>MinkNet (o)</td>
<td>102</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td>GeoMean</td>
<td>94</td>
<td>94</td>
<td>94</td>
</tr>
</tbody>
</table>

---

## Energy Saving

<table>
<thead>
<tr>
<th>Model</th>
<th>NVIDIA RTX 2080Ti</th>
<th>Intel Xeon Skylake + TPU V3</th>
<th>Intel Xeon Gold 6130</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet</td>
<td>1,319</td>
<td>1,319</td>
<td>1,319</td>
</tr>
<tr>
<td>PointNet++ (c)</td>
<td>172</td>
<td>172</td>
<td>172</td>
</tr>
<tr>
<td>PointNet++ (ps)</td>
<td>169</td>
<td>169</td>
<td>169</td>
</tr>
<tr>
<td>DGCNN</td>
<td>119</td>
<td>119</td>
<td>119</td>
</tr>
<tr>
<td>F-PointNet++</td>
<td>99</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>PointNet++ (s)</td>
<td>52</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>MinkNet (i)</td>
<td>45</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>MinkNet (o)</td>
<td>324</td>
<td>324</td>
<td>324</td>
</tr>
<tr>
<td>GeoMean</td>
<td>127</td>
<td>127</td>
<td>127</td>
</tr>
</tbody>
</table>

---

https://hanlab.mit.edu/projects/pointacc
Performance Gain over the Edge Devices

PointNet Acc. Edge Speedup

<table>
<thead>
<tr>
<th>Model</th>
<th>over NVIDIA Jetson Xavier NX</th>
<th>over NVIDIA Jetson Nano</th>
<th>over Raspberry Pi 4B</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet</td>
<td>2.2</td>
<td>6.7</td>
<td>2.2</td>
</tr>
<tr>
<td>PointNet++ (c)</td>
<td>2.3</td>
<td>7.8</td>
<td>2.3</td>
</tr>
<tr>
<td>PointNet++ (ps)</td>
<td>2.7</td>
<td>10</td>
<td>2.7</td>
</tr>
<tr>
<td>DGCNN</td>
<td>3.4</td>
<td>14</td>
<td>3.4</td>
</tr>
<tr>
<td>F-PointNet++</td>
<td>2.8</td>
<td>11</td>
<td>2.8</td>
</tr>
<tr>
<td>PointNet++ (s)</td>
<td>4.6</td>
<td>18</td>
<td>4.6</td>
</tr>
<tr>
<td>MinkNet(i)</td>
<td>2.1</td>
<td>8.3</td>
<td>2.1</td>
</tr>
<tr>
<td>MinkNet(o)</td>
<td>1.3</td>
<td>5.4</td>
<td>1.3</td>
</tr>
<tr>
<td>GeoMean</td>
<td>2.5</td>
<td>9.8</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Energy Saving

<table>
<thead>
<tr>
<th>Model</th>
<th>over NVIDIA Jetson Xavier NX</th>
<th>over NVIDIA Jetson Nano</th>
<th>over Raspberry Pi 4B</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet</td>
<td>9</td>
<td>19</td>
<td>9</td>
</tr>
<tr>
<td>PointNet++ (c)</td>
<td>11</td>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>PointNet++ (ps)</td>
<td>12</td>
<td>23</td>
<td>12</td>
</tr>
<tr>
<td>DGCNN</td>
<td>8</td>
<td>21</td>
<td>8</td>
</tr>
<tr>
<td>F-PointNet++</td>
<td>15</td>
<td>40</td>
<td>15</td>
</tr>
<tr>
<td>PointNet++ (s)</td>
<td>8</td>
<td>8.5</td>
<td>8</td>
</tr>
<tr>
<td>MinkNet(i)</td>
<td>4</td>
<td>7.2</td>
<td>4</td>
</tr>
<tr>
<td>MinkNet(o)</td>
<td>3</td>
<td>8.5</td>
<td>3</td>
</tr>
<tr>
<td>GeoMean</td>
<td>8</td>
<td>16</td>
<td>8</td>
</tr>
</tbody>
</table>

https://hanlab.mit.edu/projects/pointacc
Performance Gain of PointAcc.Edge over Mesorasi

![Graph showing speedup and energy saving gains over Mesorasi-SW on Jetson Nano and over Mesorasi-HW]

- Speedup:
  - PointNet++ (c): 10.0, 9.3, 6.2, 7.1, 4.3
  - F-PointNet++: 19.0, 21.0, GeoMean

- Energy Saving:
  - PointNet++ (c): 10, 11, 18, 15
  - F-PointNet++: 5.8, 8.7, 14, 11
  - GeoMean: 22, 28, 22, 11

[Link to project page](https://hanlab.mit.edu/projects/pointacc)
Source of Performance Gain

- Merge-sort-based implementation worsens the performance on CPU/GPU:
  - Excessive off-chip memory access between each stage of merge-sort.
  - Doubled #points in post-processing (2× #points after merge sort).
- PointAcc spatially pipelines the stages of merge sort and intersection detection.
- Using one versatile architecture for different mapping ops does not hinder the performances.
On GPU

- Fetch-on-demand flow saves the data movement cost by 3X.
- Decomposing the MM into MV multiplication significantly increases the computation overhead.

PointAcc

- Decoupled data orchestration and powerful systolic array cancel the MV overhead.
The shape of distribution of the off-chip access data size per layer is nearly the same with and without caching → caching works consistently on different layers and different datasets.

Caching reduces the off-chip memory footprint by 3.5× to 6.3×.

Temporal layer fusion cuts the off-chip memory footprint from 33% to 41%.
PointAcc.Edge v.s. Mesorasi

- Mesorasi does not support independent weights for different neighbors
- The state-of-the-art models tend to use independent weights for different neighbors
- Running the same segmentation task on S3DIS dataset, PointAcc.Edge is **9.1% higher accuracy** with **130× lower latency**.
Conclusion

- With the rise of point cloud modality, the rapid development of point cloud deep learning brings new challenges and exciting opportunities for intelligent hardware design.

https://hanlab.mit.edu/projects/pointacc
Conclusion

- With the rise of LiDAR sensors, the rapid development of point cloud deep learning brings new challenges and exciting opportunities for intelligent hardware design.

- The sparsity of point clouds in the physical space leads to two execution bottlenecks:
  - Newly introduced mapping operations for searching (input, output, weight) maps
  - Data movement overhead from gather and scatter the sparsely distributed features

https://hanlab.mit.edu/projects/pointacc
Conclusion

- With the rise of LiDAR sensors, the rapid development of point cloud deep learning brings new challenges and exciting opportunities for intelligent hardware design.

- The sparsity of point clouds in the physical space leads to two execution bottlenecks:
  - Newly introduced mapping operations for searching (input, output, weight) maps
  - Data movement overhead from gather and scatter the sparsely distributed features

- PointAcc maps diverse mapping ops into sort-based computation with one versatile architecture.

- PointAcc reduces off-chip memory access and minimize the overhead of gather and scatter by flexible caching and layer fusion.

https://hanlab.mit.edu/projects/pointacc
Conclusion

- With the rise of LiDAR sensors, the rapid development of point cloud deep learning brings new challenges and exciting opportunities for intelligent hardware design.

- The sparsity of point clouds in the physical space leads to two execution bottlenecks:
  - Newly introduced mapping operations for searching (input, output, weight) maps
  - Data movement overhead from gather and scatter the sparsely distributed features

Thank You

✉️ {yujunlin, songhan}@mit.edu
🔗 https://hanlab.mit.edu/projects/pointacc