TorchSparse: Efficient Point Cloud Inference Engine

Haotian Tang*1, Zhijian Liu*1, Xiuyu Li*2, Yujun Lin1, Song Han1

Massachusetts Institute of Technology1  University of California, Berkeley2

Project Page: https://torchsparse.mit.edu
Open Source: https://github.com/mit-han-lab/torchsparse
Point cloud data is everywhere

VR Glasses

Autonomous Driving Vehicles

AR iPhones and iPad

LiDAR Mapping Drones
3D point cloud understanding is vital in auto-driving

Efficiency is crucial for automotive applications

3D Semantic Segmentation

The trunk of an auto-driving car

Full of computers!

3D point cloud understanding is vital in auto-driving

Efficiency is crucial for automotive applications

3D sparse CNNs are not well-optimized (yet)

More accurate, less computation, but slower!

Note: Mean IoU is measured on SemanticKITTI, while latency is measured on GTX 1080Ti GPU.
Sparse convolution is tailored for point clouds

**Conventional Convolution**
- Input sparsity from ReLU
- Nonzeros will dilate

**Sparse Convolution**
- Input sparsity from the distribution in physical space
- Nonzeros will not dilate

Graham, Submanifold Sparse Convolutional Neural Networks, BMVC 2015
Sparse convolution is inefficient on GPUs

Bottleneck is data movement and mapping due to sparsity and irregularity

- Data Movement: 5%
- GEMM: 4%
- Mapping: 4%
- 2D/NMS: 5%
- Misc.: 4%

Semantic Segmentation:
- Data Movement: 44%
- GEMM: 47%

Object Detection:
- Data Movement: 23%
- GEMM: 43%
- Mapping: 15%
- 2D/NMS: 12%
Results for TorchSparse Optimizations

Trade computation for regularity and reduce memory footprint

![Graph showing performance improvements for Segmentation and Detection after optimizations.](image)
Background

Definition and existing implementation of sparse convolution
Sparse convolution computation

A sparse set of dense MMA, with rules defined by maps

Conventional Convolution

Sparse Convolution

Maps
(In, Out, Wgt)

Computation
(f_{out} = f_{out} + f_{in} \times W_{wgt}) for each entry in the maps

(P_0, Q_0, W_{1,1})

No compute
Sparse convolution computation

A sparse set of dense MMA, with rules defined by maps

**Conventional Convolution**

**Sparse Convolution**

Maps
(In, Out, Wgt)

Computation
($f_{out} = f_{out} + f_{in} \times W_{wgt}$) for each entry in the maps

(P₀, Q₀, W₁,1)
(P₀, Q₁, W₁,0)

No compute
No compute
Sparse convolution computation

A sparse set of dense MMA, with rules defined by maps

Conventional Convolution

Sparse Convolution

Maps
(In, Out, Wgt)

Computation
($f_{out} = f_{out} + f_{in} \times W_{wgt}$) for each entry in the maps

(P₀, Q₀, W₁₁)
(P₀, Q₁, W₁₀)
(P₀, Q₂, W₁₋₁)

No compute
No compute
No compute
Sparse convolution computation

A *sparse* set of *dense* MMA, with rules defined by *maps*

**Conventional Convolution**

- $f_{\text{Out}} = f_{\text{Out}} + f_{\text{In}} \times W_{\text{Wgt}}$ for each entry in the maps

**Sparse Convolution**

Maps

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

Computation

\[
\begin{align*}
\text{No compute} \\
\text{No compute} \\
\text{No compute} \\
\text{No compute}
\end{align*}
\]
Sparse convolution computation

A **sparse** set of **dense** MMA, with rules defined by **maps**

**Conventional Convolution**

**Sparse Convolution**

Maps

\[ \text{(In, Out, Wgt)} \]

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

Computation

\[ f_{\text{out}} = f_{\text{out}} + f_{\text{in}} \times W_{\text{Wgt}} \]

for each entry in the maps

No compute

No compute

No compute

\[(P_0, Q_0, W_{0,0})\]
Sparse convolution computation

A sparse set of dense MMA, with rules defined by maps

Conventional Convolution

Sparse Convolution

Maps
(In, Out, Wgt)

Computation
(f_{out} = f_{out} + f_{in} \times W_{wgt}) for each entry in the maps

(P_0, Q_0, W_{1,1})
(P_0, Q_1, W_{1,0})
(P_0, Q_2, W_{1,-1})
(P_0, Q_3, W_{0,1})
(P_0, Q_4, W_{0,0})
(P_0, Q_5, W_{0,-1})

No compute
No compute
No compute
No compute
(P_0, Q_0, W_{0,0})
No compute
Sparse convolution computation

A sparse set of dense MMA, with rules defined by maps

Conventional Convolution

Sparse Convolution

Maps
(In, Out, Wgt)

Computation
($f_{out} = f_{out} + f_{in} \times W_{wgt}$) for each entry in the maps

(P₀, Q₀, W₁,₁)
(P₀, Q₁, W₁,₀)
(P₀, Q₂, W₁,-₁)
(P₀, Q₃, W₀,₁)
(P₀, Q₄, W₀,₀)
(P₀, Q₅, W₀,-₁)
(P₀, Q₈, W₁,-₁)

No compute
No compute
No compute
No compute
(P₀, Q₀, W₀,₀)
No compute
No compute
Sparse convolution computation

A sparse set of dense MMA, with rules defined by maps

Conventional Convolution

Sparse Convolution

Maps
(In, Out, Wgt)

Computation
($f_{out} = f_{out} + f_{in} \times W_{wgt}$) for each entry in the maps

(P₀, Q₀, W₁₁)
(P₀, Q₁, W₁₀)
(P₀, Q₂, W₁₋₁)
(P₀, Q₃, W₀₁)
(P₀, Q₄, W₀₀)
(P₀, Q₅, W₀₋₁)
(P₀, Q₈, W₋₁₁)
(P₀, Q₉, W₋₁₀)

No compute
No compute
No compute
No compute
No compute
No compute
No compute
No compute
Sparse convolution computation

A *sparse* set of *dense* MMA, with rules defined by *maps*

Conventional Convolution

Sparse Convolution

Maps

(\text{In}, \text{Out}, \text{Wgt})

\begin{align*}
(P_0, Q_0, W_{1,1}) \\
(P_0, Q_1, W_{1,0}) \\
(P_0, Q_2, W_{1,-1}) \\
(P_0, Q_3, W_{0,1}) \\
(P_0, Q_4, W_{0,0}) \\
(P_0, Q_5, W_{0,-1}) \\
(P_0, Q_8, W_{-1,1}) \\
(P_0, Q_9, W_{-1,0}) \\
(P_0, Q_{10}, W_{-1,-1})
\end{align*}

Computation

\((f_{\text{out}} = f_{\text{out}} + f_{\text{in}} \times W_{\text{Wgt}})\) for each entry in the maps

\begin{align*}
\text{No compute} \\
\text{No compute} \\
\text{No compute} \\
\text{No compute} \\
\text{No compute} \\
(P_0, Q_0, W_{0,0}) \\
\text{No compute} \\
\text{No compute} \\
(P_0, Q_1, W_{-1,-1})
\end{align*}

9 matrix multiplications

2 matrix multiplications
Existing GPU implementation of sparse convolution

Weight-stationary computation, separate matmul for different weights
Existing GPU implementation of sparse convolution

Weight-stationary computation, separate matmul for different weights

$\text{Input Features} \rightarrow \text{Input Buffer} \rightarrow \text{Weight} \rightarrow \text{Partial Sum} \rightarrow \text{Output Features}$

$\text{Maps} (\text{In}, \text{Out}, \text{Wgt})$

$(P_0, Q_1, W_{-1,-1})$
$(P_3, Q_4, W_{-1,-1})$
$(P_1, Q_3, W_{-1,0})$
$(P_0, Q_0, W_{0,0})$
$(P_1, Q_1, W_{0,0})$
$(P_2, Q_2, W_{0,0})$
$(P_3, Q_3, W_{0,0})$
$(P_4, Q_4, W_{0,0})$
$(P_3, Q_1, W_{1,0})$
$(P_1, Q_0, W_{1,1})$
$(P_4, Q_2, W_{1,1})$

$5 \times C_{\text{in}}$
$1 \times C_{\text{in}}$
$C_{\text{in}} \times C_{\text{out}}$
$1 \times C_{\text{out}}$

$C_{\text{out}} \times 5$

$W_{-1,0}$

$\text{PSUM3}$

$f_3 = f_3 + f_1 \times W_{-1,0}$
Existing GPU implementation of sparse convolution

Weight-stationary computation, separate matmul for different weights

\[
\begin{align*}
W & = W_{0,0} \\
\end{align*}
\]

Note: maps for \( W_{0,0} \) contains all entries.
Existing GPU implementation of sparse convolution

Weight-stationary computation, separate matmul for different weights

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

Workload

Input Features | Input Buffer | Weight | Partial Sum | Output Features
--- | --- | --- | --- | ---
P₀ | 1 × Cᵢⁿ | | PSUM₁ | Q₀
P₁ | | | | Q₁
P₂ | | | | Q₂
P₃ | | W₁,₀ | | Q₃
P₄ | | | | Q₄

f₁ = f₁ + f₃ × W₁,₀
Existing GPU implementation of sparse convolution

Weight-stationary computation, separate matmul for different weights

\[
\begin{align*}
    f_0 &= f_0 + f_1 \times W_{1,1} \\
    f_3 &= f_3 + f_4 \times W_{1,1}
\end{align*}
\]
TorchSparse Overview

System, API and optimization overview
TorchSparse system overview

SparseConvNets: MinkUNet, CenterPoint, ...

TorchSparse APIs:
spnn.Conv3d, spF.conv3d...

Output Coord. Construction → Map Search

Fused Locality-Aware Scatter/Gather

Adaptively Grouped MatMul

PyTorch CUDA Extension
TorchSparse has PyTorch-like APIs

```python
import torch.nn as nn
class ConvBlock(nn.Sequential):
    def __init__(self,
        in_channels: int,
        out_channels: int,
        kernel_size: int,
        stride: Union[int, list, tuple] = 1,
        dilation: int = 1) -> None:
        super().__init__(nn.Conv2d(in_channels, out_channels, kernel_size, stride=stride, dilation=dilation),
                         nn.BatchNorm2d(out_channels),
                         nn.ReLU(True))

import torchsparse.nn as spnn
class SparseConvBlock(nn.Sequential):
    def __init__(self,
        in_channels: int,
        out_channels: int,
        kernel_size: int,
        stride: Union[int, list, tuple] = 1,
        dilation: int = 1) -> None:
        super().__init__(spnn.Conv3d(in_channels, out_channels, kernel_size, stride=stride, dilation=dilation),
                         spnn.BatchNorm(out_channels),
                         spnn.ReLU(True))
```
TorchSparse optimization overview

Locality-Aware Access

Adaptive Grouping

Locality-Aware Access

Matrix-Matrix Multiplication

Gather

Scatter-Accumulate

Gather

X

F0
F1
F2
F3
F4

Input
Features

Apply BMM

W_{0,0} = \text{PSUM 0}
PSUM 1
PSUM 2
PSUM 3
PSUM 4

W_{1,1} = \text{PSUM 1}
PSUM 3
PSUM 4

W_{-1,1} = \text{PSUM 4}
PSUM 3
PSUM 2

W_{-1,0} = \text{PSUM 3}
PSUM 2
PSUM 1

W_{1,0} = \text{PSUM 1}
PSUM 0

Apply MM

Scatter

X

F0
F1
F2
F3
F4

Output
Features

Locality-Aware Access

Adaptive Grouping

Locality-Aware Access

TorchSparse: Efficient Point Cloud Inference Engine

torchsparse.mit.edu

Haotian Tang*, Zhijian Liu*, Xiuyu Li*, Yujun Lin and Song Han
Trading computation for regularity

Optimizing matrix multiplication via adaptive grouping

Matrix-Matrix Multiplication

\[
\begin{align*}
F_0 \times W_{-1,-1} &= \text{PSUM 1, PSUM 4} \\
F_3 \times W_{-1,0} &= \text{PSUM 3} \\
F_1 \times W_{1,0} &= \text{PSUM 1} \\
F_3 \times W_{1,1} &= \text{PSUM 0, PSUM 3} \\
F_0 \times W_{0,0} &= \text{PSUM 0, PSUM 1, PSUM 2, PSUM 3, PSUM 4}
\end{align*}
\]
Reducing memory footprint

Optimizing data movement with fused and locality-aware memory access
Trading computation for regularity

Matrix multiplication optimizations
Trading computation for regularity

Separate computation (baseline): many kernel calls, low device utilization

Separate Computation

Worst

Best

Computation overhead

Computation regularity
Trading computation for regularity

Dense convolution: best regularity but large computation overhead
Trading computation for regularity
Computing with grouping: balancing overhead and regularity

Separate Computation

Dense Convolution

Computation with grouping

Worst

Best

Computation overhead

Computation regularity

Computation overhead

Computation regularity

Computation overhead

Computation regularity

Extra computation = 2 / 28
(Small overhead)
Trading computation for regularity

Searching customized strategy for different model and datasets

Increasing **regularity** helps improve latency
Padding overhead hurts latency

---

### Speedup Over Baseline vs. Number of Groups

<table>
<thead>
<tr>
<th>Number of Groups</th>
<th>Speedup Over Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>1.6</td>
</tr>
<tr>
<td>24</td>
<td>1.2</td>
</tr>
<tr>
<td>22</td>
<td>0.8</td>
</tr>
<tr>
<td>20</td>
<td>0.4</td>
</tr>
<tr>
<td>18</td>
<td>1.8</td>
</tr>
<tr>
<td>16</td>
<td>1.2</td>
</tr>
<tr>
<td>14</td>
<td>0.8</td>
</tr>
<tr>
<td>12</td>
<td>0.4</td>
</tr>
<tr>
<td>10</td>
<td>1.8</td>
</tr>
<tr>
<td>8</td>
<td>1.2</td>
</tr>
<tr>
<td>6</td>
<td>0.8</td>
</tr>
<tr>
<td>4</td>
<td>0.4</td>
</tr>
<tr>
<td>2</td>
<td>1.8</td>
</tr>
<tr>
<td>0</td>
<td>1.2</td>
</tr>
</tbody>
</table>

---

### Map Size vs. Weight Index

**SemanticKITTI**

- Weight Index: 1, 4, 7, 10, 13, 16, 19, 22, 2527
- Map Size: $10^2$, $10^3$, $10^4$, $10^5$

**nuScenes**

- Weight Index: 1, 4, 7, 10, 13, 16, 19, 22, 2527
- Map Size: $10^2$, $10^3$, $10^4$, $10^5$
Results on matrix multiplication optimizations

SemanticKITTI

<table>
<thead>
<tr>
<th>TFLOP/s</th>
<th>Baseline</th>
<th>Fixed Grouping</th>
<th>Adaptive Grouping</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>8.7</td>
<td>11.9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Normalized Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
</tr>
</tbody>
</table>
Results on matrix multiplication optimizations

nuScenes: fixed grouping has best TFLOP/s but adaptive grouping is faster

This is because fixed grouping introduced large amount of redundant computation.
Reducing memory footprint

Data movement optimizations
Recall: data movement is a major overhead

43-47% of total runtime for 3D perception models

Data Movement: 47%
GEMM: 44%
Mapping: 4%
2D/NMS: 5%
Misc.: 4%

Semantic Segmentation
Object Detection
**Quantized and vectorized memory access**

<table>
<thead>
<tr>
<th>Channels (4B)</th>
<th>$c_0$</th>
<th>$c_1$</th>
<th>...</th>
<th>$c_{30}$</th>
<th>$c_{31}$</th>
<th>$c_{32}$</th>
<th>$c_{33}$</th>
<th>...</th>
<th>$c_{62}$</th>
<th>$c_{63}$</th>
<th>...</th>
<th>$c_{255}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threads</td>
<td>$t_0$</td>
<td>$t_1$</td>
<td>...</td>
<td>$t_{30}$</td>
<td>$t_{31}$</td>
<td>$t_{32}$</td>
<td>$t_{33}$</td>
<td>...</td>
<td>$t_{60}$</td>
<td>$t_{61}$</td>
<td>...</td>
<td>$t_{255}$</td>
</tr>
<tr>
<td>Warp (128B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Warp #0</td>
<td>Warp #1</td>
<td>Warp #7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Original FP32 access (8 warps for 256 channels)

<table>
<thead>
<tr>
<th>Channels (2B)</th>
<th>$c_0$</th>
<th>$c_1$</th>
<th>$c_2$</th>
<th>...</th>
<th>$c_{60}$</th>
<th>$c_{61}$</th>
<th>$c_{62}$</th>
<th>$c_{63}$</th>
<th>...</th>
<th>$c_{124}$</th>
<th>$c_{125}$</th>
<th>$c_{126}$</th>
<th>$c_{127}$</th>
<th>...</th>
<th>$c_{254}$</th>
<th>$c_{255}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threads</td>
<td>$t_0$</td>
<td>$t_1$</td>
<td>...</td>
<td>$t_{30}$</td>
<td>$t_{31}$</td>
<td>$t_{32}$</td>
<td>$t_{33}$</td>
<td>...</td>
<td>$t_{90}$</td>
<td>$t_{91}$</td>
<td>...</td>
<td>$t_{255}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warp (128B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Warp #0</td>
<td>Warp #1</td>
<td>Warp #3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Optimized FP16 access (4 warps for 256 channels)

**Vectorized** and **quantized** memory access reduces the memory footprint by 2x.
Weight-stationary scatter-gather: not cache friendly

Cache Hit

\( (P_0, Q_1) \)
\( (P_3, Q_4) \)
\( \vdots \)
\( (P_{95029}, Q_{95133}) \)
\( (P_{95077}, Q_{95181}) \)
\( W_{-1,-1,-1} \)

Cache Miss

\( (P_1, Q_2) \)
\( (P_4, Q_3) \)
\( \vdots \)
\( (P_{95133}, Q_{95029}) \)
\( (P_{95180}, Q_{95077}) \)
\( W_{1,1,1} \)
Weight-stationary scatter-gather: not cache friendly

Gather phase for weight $W_{-1,-1,-1}$

Cache Hit

Cache Miss

$(P_0, Q_1)$

$W_{-1,-1,-1}$  $W_{-1,-1,0}$  $W_{1,1,0}$  $W_{1,1,1}$
Weight-stationary scatter-gather: not cache friendly

Gather phase for weight $W_{-1,-1,-1}$

Cache Hit

$(P_0, Q_1)$

$(P_3, Q_4)$

Cache Miss

$W_{-1,-1,-1}$   $W_{-1,-1,0}$   $W_{1,1,0}$   $W_{1,1,1}$
Weight-stationary scatter-gather: not cache friendly

Gather phase for weight $W_{-1,-1,-1}$

Because the maps for $W_{-1,-1,-1}$ are unique, there is no cache reuse during the gather phase for $W_{-1,-1,-1}$.
Weight-stationary scatter-gather: not cache friendly

Scatter phase for weight $W_{-1,-1,-1}$

- $(P_0, Q_1)$
- $(P_3, Q_4)$
- $(P_{95029}, Q_{95133})$
- $(P_{95077}, Q_{95181})$

- $W_{-1,-1,-1}$
- $W_{-1,-1,0}$
- $W_{1,1,0}$
- $W_{1,1,1}$
Weight-stationary scatter-gather: not cache friendly

Scatter phase for weight $W_{-1,-1,-1}$

Cache Hit

Cache Miss

$((P_0, Q_1), W_{-1,-1,-1})$

$((P_3, Q_4), W_{-1,-1,-1})$

$\ldots$

$((P_{95029}, Q_{95133}), W_{1,1,0})$

$((P_{95077}, Q_{95181}), W_{1,1,1})$
Weight-stationary scatter-gather: not cache friendly

Scatter phase for weight $W_{-1,-1,-1}$

Because the maps for $W_{-1,-1,-1}$ are unique, there is no cache reuse during the scatter phase for $W_{-1,-1,-1}$.
Weight-stationary scatter-gather: not cache friendly

It is also the case for all other weights because the cache is not large enough to hold all input features / output partial sums.
Solution: fused and locality-aware scatter-gather

Changing the map layout

**Weight-stationary:**
vertical traversal
For each weight, what are the (input, output) pairs using this weight?

**Input/Output-stationary:**
horizontal traversal
For each input/output, what weight sub-map does it appear in?
Solution: fused and locality-aware scatter-gather

Gather phase for all input points

Input-stationary gather

First accesses to all input points are mandatory cache misses.
Solution: fused and locality-aware scatter-gather

Gather phase for all input points

Input-stationary gather

All subsequent accesses hits because the loaded input features are stored in the register file.
Solution: fused and locality-aware scatter-gather

Scatter phase for all output points

Input-stationary gather

Output-stationary scatter

The situation is similar for output-stationary scatter.
Solution: fused and locality-aware scatter-gather

Improving the cache hit ratio via reordering memory accesses

(a) baseline: weight-stationary scatter-gather

(b) ours: fused and locality-aware scatter-gather
Evaluation
## Results

TorchSparse achieves consistent improvements on different GPUs

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Implementation</th>
<th>RTX 3090</th>
</tr>
</thead>
<tbody>
<tr>
<td>SK-MinkUNet (1.0x)</td>
<td>Baseline Implementation</td>
<td>0.58</td>
</tr>
<tr>
<td>SK-MinkUNet (0.5x)</td>
<td>Baseline Implementation</td>
<td>0.47</td>
</tr>
<tr>
<td>NS-MinkUNet (3f)</td>
<td>Baseline Implementation</td>
<td>0.50</td>
</tr>
<tr>
<td>NS-MinkUNet (1f)</td>
<td>Baseline Implementation</td>
<td>0.55</td>
</tr>
<tr>
<td>NS-CenterPoint (10f)</td>
<td>Baseline Implementation</td>
<td>0.73</td>
</tr>
<tr>
<td>WM-CenterPoint (3f)</td>
<td>Baseline Implementation</td>
<td>0.57</td>
</tr>
<tr>
<td>WM-CenterPoint (1f)</td>
<td>Baseline Implementation</td>
<td>0.54</td>
</tr>
<tr>
<td>Geomean</td>
<td>Baseline Implementation</td>
<td>0.58</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Implementation</th>
<th>RTX 3090</th>
</tr>
</thead>
<tbody>
<tr>
<td>SK-MinkUNet (1.0x)</td>
<td>MinkowskiEngine 0.5.4</td>
<td>0.66</td>
</tr>
<tr>
<td>SK-MinkUNet (0.5x)</td>
<td>MinkowskiEngine 0.5.4</td>
<td>0.50</td>
</tr>
<tr>
<td>NS-MinkUNet (3f)</td>
<td>MinkowskiEngine 0.5.4</td>
<td>0.65</td>
</tr>
<tr>
<td>NS-MinkUNet (1f)</td>
<td>MinkowskiEngine 0.5.4</td>
<td>0.73</td>
</tr>
<tr>
<td>NS-CenterPoint (10f)</td>
<td>MinkowskiEngine 0.5.4</td>
<td>0.55</td>
</tr>
<tr>
<td>WM-CenterPoint (3f)</td>
<td>MinkowskiEngine 0.5.4</td>
<td>0.75</td>
</tr>
<tr>
<td>WM-CenterPoint (1f)</td>
<td>MinkowskiEngine 0.5.4</td>
<td>0.65</td>
</tr>
<tr>
<td>Geomean</td>
<td>MinkowskiEngine 0.5.4</td>
<td>0.60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Implementation</th>
<th>RTX 3090</th>
</tr>
</thead>
<tbody>
<tr>
<td>SK-MinkUNet (1.0x)</td>
<td>SPConv 1.2.1 (FP32)</td>
<td>0.64</td>
</tr>
<tr>
<td>SK-MinkUNet (0.5x)</td>
<td>SPConv 1.2.1 (FP32)</td>
<td>0.50</td>
</tr>
<tr>
<td>NS-MinkUNet (3f)</td>
<td>SPConv 1.2.1 (FP32)</td>
<td>0.65</td>
</tr>
<tr>
<td>NS-MinkUNet (1f)</td>
<td>SPConv 1.2.1 (FP32)</td>
<td>0.73</td>
</tr>
<tr>
<td>NS-CenterPoint (10f)</td>
<td>SPConv 1.2.1 (FP32)</td>
<td>0.55</td>
</tr>
<tr>
<td>WM-CenterPoint (3f)</td>
<td>SPConv 1.2.1 (FP32)</td>
<td>0.75</td>
</tr>
<tr>
<td>WM-CenterPoint (1f)</td>
<td>SPConv 1.2.1 (FP32)</td>
<td>0.65</td>
</tr>
<tr>
<td>Geomean</td>
<td>SPConv 1.2.1 (FP32)</td>
<td>0.65</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Implementation</th>
<th>RTX 3090</th>
</tr>
</thead>
<tbody>
<tr>
<td>SK-MinkUNet (1.0x)</td>
<td>SPConv 1.2.1 (FP16)</td>
<td>0.73</td>
</tr>
<tr>
<td>SK-MinkUNet (0.5x)</td>
<td>SPConv 1.2.1 (FP16)</td>
<td>0.50</td>
</tr>
<tr>
<td>NS-MinkUNet (3f)</td>
<td>SPConv 1.2.1 (FP16)</td>
<td>0.75</td>
</tr>
<tr>
<td>NS-MinkUNet (1f)</td>
<td>SPConv 1.2.1 (FP16)</td>
<td>0.65</td>
</tr>
<tr>
<td>NS-CenterPoint (10f)</td>
<td>SPConv 1.2.1 (FP16)</td>
<td>0.75</td>
</tr>
<tr>
<td>WM-CenterPoint (3f)</td>
<td>SPConv 1.2.1 (FP16)</td>
<td>0.65</td>
</tr>
<tr>
<td>WM-CenterPoint (1f)</td>
<td>SPConv 1.2.1 (FP16)</td>
<td>0.65</td>
</tr>
<tr>
<td>Geomean</td>
<td>SPConv 1.2.1 (FP16)</td>
<td>0.65</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Implementation</th>
<th>RTX 3090</th>
</tr>
</thead>
<tbody>
<tr>
<td>SK-MinkUNet (1.0x)</td>
<td>TorchSparse</td>
<td>1.00</td>
</tr>
<tr>
<td>SK-MinkUNet (0.5x)</td>
<td>TorchSparse</td>
<td>1.00</td>
</tr>
<tr>
<td>NS-MinkUNet (3f)</td>
<td>TorchSparse</td>
<td>1.00</td>
</tr>
<tr>
<td>NS-MinkUNet (1f)</td>
<td>TorchSparse</td>
<td>1.00</td>
</tr>
<tr>
<td>NS-CenterPoint (10f)</td>
<td>TorchSparse</td>
<td>1.00</td>
</tr>
<tr>
<td>WM-CenterPoint (3f)</td>
<td>TorchSparse</td>
<td>1.00</td>
</tr>
<tr>
<td>WM-CenterPoint (1f)</td>
<td>TorchSparse</td>
<td>1.00</td>
</tr>
<tr>
<td>Geomean</td>
<td>TorchSparse</td>
<td>1.00</td>
</tr>
</tbody>
</table>

---

**Baseline Implementation**
- MinkowskiEngine 0.5.4
- SPConv 1.2.1 (FP32)
- SPConv 1.2.1 (FP16)
- TorchSparse

**Datasets**
- **SK**: SemanticKITTI (ICCV’19)
- **NS**: nuScenes (CVPR’20)
- **WM**: Waymo (CVPR’20)
TorchSparse achieves consistent improvements on different GPUs

SK: SemanticKITTI (ICCV’19), NS: nuScenes (CVPR’20), WM: Waymo (CVPR’20)
Results

TorchSparse achieves consistent improvements on different GPUs

<table>
<thead>
<tr>
<th>Baseline Implementation</th>
<th>MinkowskiEngine 0.5.4</th>
<th>SPConv 1.2.1 (FP32)</th>
<th>SPConv 1.2.1 (FP16)</th>
<th>TorchSparse</th>
</tr>
</thead>
</table>

**RTX 3090**

<table>
<thead>
<tr>
<th>SK-MinkUNet (1.0x)</th>
<th>SK-MinkUNet (0.5x)</th>
<th>NS-MinkUNet (3f)</th>
<th>NS-MinkUNet (1f)</th>
<th>NS-CenterPoint (10f)</th>
<th>WM-CenterPoint (3f)</th>
<th>WM-CenterPoint (1f)</th>
<th>Geomean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.58</td>
<td>0.60</td>
<td>0.50</td>
<td>0.65</td>
<td>0.73</td>
<td>0.55</td>
<td>0.67</td>
<td>0.06</td>
</tr>
<tr>
<td>0.47</td>
<td>0.67</td>
<td>0.50</td>
<td>0.55</td>
<td>0.67</td>
<td>0.55</td>
<td>0.67</td>
<td>0.06</td>
</tr>
<tr>
<td>0.53</td>
<td>0.67</td>
<td>0.50</td>
<td>0.67</td>
<td>0.55</td>
<td>0.67</td>
<td>0.67</td>
<td>0.06</td>
</tr>
</tbody>
</table>

**RTX 2080Ti**

<table>
<thead>
<tr>
<th>SK-MinkUNet (1.0x)</th>
<th>SK-MinkUNet (0.5x)</th>
<th>NS-MinkUNet (3f)</th>
<th>NS-MinkUNet (1f)</th>
<th>NS-CenterPoint (10f)</th>
<th>WM-CenterPoint (3f)</th>
<th>WM-CenterPoint (1f)</th>
<th>Geomean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.52</td>
<td>0.60</td>
<td>0.50</td>
<td>0.65</td>
<td>0.73</td>
<td>0.55</td>
<td>0.67</td>
<td>0.06</td>
</tr>
<tr>
<td>0.46</td>
<td>0.67</td>
<td>0.50</td>
<td>0.55</td>
<td>0.67</td>
<td>0.55</td>
<td>0.67</td>
<td>0.06</td>
</tr>
<tr>
<td>0.69</td>
<td>0.67</td>
<td>0.50</td>
<td>0.67</td>
<td>0.55</td>
<td>0.67</td>
<td>0.67</td>
<td>0.06</td>
</tr>
</tbody>
</table>

**RTX 1080Ti**

<table>
<thead>
<tr>
<th>SK-MinkUNet (1.0x)</th>
<th>SK-MinkUNet (0.5x)</th>
<th>NS-MinkUNet (3f)</th>
<th>NS-MinkUNet (1f)</th>
<th>NS-CenterPoint (10f)</th>
<th>WM-CenterPoint (3f)</th>
<th>WM-CenterPoint (1f)</th>
<th>Geomean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.68</td>
<td>0.60</td>
<td>0.50</td>
<td>0.65</td>
<td>0.73</td>
<td>0.55</td>
<td>0.67</td>
<td>0.06</td>
</tr>
<tr>
<td>0.61</td>
<td>0.67</td>
<td>0.50</td>
<td>0.55</td>
<td>0.67</td>
<td>0.55</td>
<td>0.67</td>
<td>0.06</td>
</tr>
<tr>
<td>0.85</td>
<td>0.67</td>
<td>0.50</td>
<td>0.67</td>
<td>0.55</td>
<td>0.67</td>
<td>0.67</td>
<td>0.06</td>
</tr>
</tbody>
</table>

SK: SemanticKITTI (ICCV’19), NS: nuScenes (CVPR’20), WM: Waymo (CVPR’20)
Conclusion

Sparse convolution is an emerging operator in point cloud processing. We trade computation for regularity, optimizing matrix multiplication in sparse convolution via adaptive grouping. We reduce the memory footprint of sparse convolution via fused and locality-aware memory access.

🔗 https://torchsparse.mit.edu
🔗 https://github.com/mit-han-lab/torchsparse
Conclusion

Sparse convolution is an emerging operator in point cloud processing.

We trade computation for regularity, optimizing matrix multiplication in sparse convolution via adaptive grouping.

We reduce the memory footprint of sparse convolution via fused and locality-aware memory access.

🔗 https://torchsparse.mit.edu

🔗 https://github.com/mit-han-lab/torchsparse
Related work

Full-stack efficient 3D perception for auto-driving

Lidar and 3D point clouds: **sparse, irregular**, and **large** memory footprint => hardware unfriendly
Camera and 2D images: **high resolution, multi-camera, real-time** => computationally hungry
Sensor fusion: multiple sensors, multiple tasks => even more computationally hungry

---

**Algorithm**
- 3D Light-weight Neural-net [PVCNN, NeurIPS’19 Spotlight]
- 3D Neural Architecture Search [SPVNAS, ECCV’20]
- Multi-Task Multi Sensor Fusion [BEVFusion, arXiv’22]

**Software**
- 3D Inference Engine on GPU
  [TorchSparse, MLSys’22]

**Hardware**
- 3D Hardware Accelerator
  [PointAcc, MICRO’21]

---

Video credit: [https://semantic-kitti.org](https://semantic-kitti.org)
Sparse convolution is an emerging operator in point cloud processing. We trade computation for regularity, optimizing matrix multiplication in sparse convolution via adaptive grouping. We reduce the memory footprint of sparse convolution via fused and locality-aware memory access.

🔗 https://torchsparse.mit.edu
🔗 https://github.com/mit-han-lab/torchsparse

Thank you!