SparseTrain: Leveraging Dynamic Sparsity in Software for Training DNNs on General-Purpose SIMD Processors

Zhangxiaowen Gong, Houxiang Ji, Christopher W. Fletcher
Christopher J. Hughes*, Josep Torrellas

University of Illinois at Urbana-Champaign, *Intel Labs

PACT 2020
Sparsity and Work Skipping

- Sparsity: zeros inside a data structure
- Multiply-Accumulate Operation (MAC): \( c = c + a \times b \)
- Zero in multiplicands: skippable!

\[ c + 0 \times b = c \]
Dynamic Sparsity in DNNs

- Zeros appear dynamically during training/inference
- Main source: ReLU
  - Activates the neurons of a DNN layer
  - Produces 40-90% sparsity
- Hard to exploit
  - Unstructured
  - Changes over time
  - Moderate sparsity level

\[
f(x) = \max(0, x)
\]

\[
f'(x) = \begin{cases} 
1, & \text{if } x > 0 \\
0, & \text{otherwise}
\end{cases}
\]
Contribution: SparseTrain

- The first software-only algorithm to speedup DNN training by exploiting dynamic sparsity
  - Skips computation at runtime
  - Uses a regular (dense) representation
  - Generates JIT kernels
  - Vectorized

- Applicable to all training phases
  - Forward propagation *(same as inference)*
  - Backward input propagation
  - Backward weight propagation
SparseTrain for Convolutional Neural Networks (CNNs)

- SparseTrain is applicable to GEMM-like computation; we focus on CNNs.
- Convolutional layers:
  - High compute-to-memory ratio
  - Most time consuming component
  - Usually activated by ReLU
- Accelerates training by 1.3x-2.2x and inference by 1.4x-1.9x
- In this talk, we present forward propagation
  - Backward input/weight propagations are similar
Forward Convolution

\[ Y_{i,k,x,y} = \sum_{c=0}^{C-1} \sum_{u=0}^{R-1} \sum_{v=0}^{S-1} I_{i,c,x+u,y+v} \times G_{k,c,u,v} \]

\[ i \in N \quad k \in K \quad c \in C \quad x \in W \quad u \in R \quad y \in H \quad v \in S \]

Input Weights Output
Forward Convolution

\[ Y_{i,k,x,y} = \sum_{c=0}^{C-1} \sum_{u=0}^{R-1} \sum_{v=0}^{S-1} D_{i,c,x+u,y+v} \times G_{k,c,u,v} \]

\[ i \in N \quad k \in K \quad c \in C \]
\[ y \in H \quad x \in W \quad u \in R \]
Forward Convolution

\[ Y_{i,k,x,y} = \sum_{c=0}^{C-1} \sum_{u=0}^{R-1} \sum_{v=0}^{S-1} D_{i,c,x+u,y+v} \times G_{k,c,u,v} \quad i \in N \quad k \in K \quad c \in C \]

\[ Y \quad 1 \quad K \]

Input

Weights

Output
Forward Convolution

\[ Y_{i,k,x,y} = \sum_{c=0}^{C-1} \sum_{u=0}^{R-1} \sum_{v=0}^{S-1} D_{i,c,x+u,y+v} \times G_{k,c,u,v} \]

Input & Weights & Output

\[ i \in N \quad k \in K \quad c \in C \]
\[ y \in H \quad x \in W \quad u \in R \]
\[ v \in S \]
Forward Convolution

\[ Y_{i,k,x,y} = \sum_{c=0}^{C-1} \sum_{u=0}^{R-1} \sum_{v=0}^{S-1} D_{i,c,x+u,y+v} \times G_{k,c,u,v} \]

\[ i \in N \quad k \in K \quad c \in C \]
\[ y \in H \quad x \in W \quad u \in R \]
\[ v \in S \]
Forward Convolution

\[ Y_{i,k,x,y} = \sum_{c=0}^{C-1} \sum_{u=0}^{R-1} \sum_{v=0}^{S-1} D_{i,c,x+u,y+v} \times G_{k,c,u,v} \]

\( i \in N \quad k \in K \quad c \in C \\
\( y \in H \quad x \in W \quad u \in R \\
\( v \in S \)
Naïve Sparse Forward Convolution

\[ Y_{i,k,x,y} = \sum_{c=0}^{C-1} \sum_{u=0}^{R-1} \sum_{v=0}^{S-1} D_{i,c,x+u,y+v} \times G_{k,c,u,v} \]

for \( i \) in 0 to N-1
for \( k \) in 0 to K-1
for \( y \) in 0 to H-1
for \( x \) in 0 to W-1
for \( c \) in 0 to C-1
for \( v \) in 0 to S-1
for \( u \) in 0 to R-1
\[ Y_{i,k,x,y} = D_{i,c,x+u,y+v} \times G_{k,c,u,v} \]

stationary
Naïve Sparse Forward Convolution

- Step 1: permute the computation loop nest to station the input

```plaintext
for i in 0 to N-1
    for k in 0 to K-1
        for y in 0 to H-1
            for x in 0 to W-1
                for c in 0 to C-1
                    for v in 0 to S-1
                        for u in 0 to R-1
                            Y_{i,k,x,y} += D_{i,c,x+u,y+v} \cdot G_{k,c,u,v}
```

- For the transformed step:

```plaintext
for i in 0 to N-1
    for c in 0 to C-1
        for y in S-1 to H+S-2
            for x in R-1 to W+R-2
                for k in 0 to K-1
                    for v in 0 to S-1
                        for u in 0 to R-1
                            Y_{i,k,x-u,y-v} += D_{i,c,x,y} \cdot G_{k,c,u,v}
```

- For the stationary step:

```plaintext
for k in 0 to K-1
    for v in 0 to S-1
        for u in 0 to R-1
            Y_{i,k,x-u,y-v} += D_{i,c,x,y} \cdot G_{k,c,u,v}
```
Naïve Sparse Forward Convolution

- Step 1: permute the computation loop nest to station the input
- Step 2: check the input for zero and skip compute accordingly

\[
\text{for } i \text{ in } 0 \text{ to } N-1 \\
\text{for } k \text{ in } 0 \text{ to } K-1 \\
\text{for } y \text{ in } 0 \text{ to } H-1 \\
\text{for } x \text{ in } 0 \text{ to } W-1 \\
\text{for } c \text{ in } 0 \text{ to } C-1 \\
\text{for } v \text{ in } 0 \text{ to } S-1 \\
\text{for } u \text{ in } 0 \text{ to } R-1 \\
Y_{i,k,x,y} = D_{i,c,x+u,y+v} G_{k,c,u,v}
\]

\[
\text{transform} \\
\text{for } i \text{ in } 0 \text{ to } N-1 \\
\text{for } c \text{ in } 0 \text{ to } C-1 \\
\text{for } y \text{ in } S-1 \text{ to } H+S-2 \\
\text{for } x \text{ in } R-1 \text{ to } W+R-2 \\
\text{for } k \text{ in } 0 \text{ to } K-1 \\
\text{if } D_{i,c,x,y} \neq 0 \\
\text{for } v \text{ in } 0 \text{ to } S-1 \\
\text{for } u \text{ in } 0 \text{ to } R-1 \\
Y_{i, k, x-u, y-v} = D_{i,c,x,y} G_{k,c,u,v}
\]
Naïve Sparse Forward Convolution

- Step 1: permute the computation loop nest to station the input
- Step 2: check the input for zero and skip compute accordingly
- Step 3: vectorize the compute along the output channel dimension $K$

```
for $i$ in 0 to $N-1$
  for $k$ in 0 to $K-1$
    for $y$ in 0 to $H-1$
      for $x$ in 0 to $W-1$
        for $c$ in 0 to $C-1$
          for $v$ in 0 to $S-1$
            for $u$ in 0 to $R-1$
              $Y_{i,k,x,y} += D_{i,c,x+u,y+v}G_{k,c,u,v}$

for $i$ in 0 to $N-1$
  for $c$ in 0 to $C-1$
    for $y$ in $S-1$ to $H+S-2$
      for $x$ in $R-1$ to $W+R-2$
        if $D_{i,c,x,y} \neq 0$
          for $k$ in 0 to $K-V$ step $V$
            for $v$ in 0 to $S-1$
              for $u$ in 0 to $R-1$
                $Y_{i,[k:k+V-1],x-u,y-v} += D_{i,c,x,y}G_{[k:k+V-1],c,u,v}$
```
Naïve Sparse Forward Convolution: Problems

- Hard to parallelize

for \( i \) in 0 to \( N-1 \) \textbf{in parallel}
  for \( c \) in 0 to \( C-1 \)
    for \( y \) in \( S-1 \) to \( H+S-2 \)
      for \( x \) in \( R-1 \) to \( W+R-2 \)
        if \( D_{i,c,x,y} \neq 0 \)
          for \( k \) in 0 to \( K-V \) step \( V \)
            for \( v \) in 0 to \( S-1 \)
              for \( u \) in 0 to \( R-1 \)
                \( Y_{i,[k:k+V-1],x-u,y-v} += D_{i,c,x,y} G_{[k:k+V-1],c,u,v} \)
Naïve Sparse Forward Convolution: Problems

- Hard to parallelize
- Register spilling

```plaintext
for i in 0 to N-1
    for c in 0 to C-1
        for y in S-1 to H+S-2
            for x in R-1 to W+R-2
                if D_{i,c,x,y} ≠ 0
                    for k in 0 to K-V step V
                        for v in 0 to S-1
                            for u in 0 to R-1
                                Yi, [k:k+V-1], x-u, y-v
                                += D_{i,c,x,y}^G [k:k+V-1], c, u, v
```
Naïve Sparse Forward Convolution: Problems

- Hard to parallelize
- Register spilling
- Branch misprediction

```plaintext
for i in 0 to N-1
  for c in 0 to C-1
    for y in S-1 to H+S-2
      for x in R-1 to W+R-2
        if D_{i,c,x,y} ≠ 0
          for k in 0 to K-V step V
            for v in 0 to S-1
              for u in 0 to R-1
                Y_{i,[k:k+V-1],x-u,y-v}
                += D_{i,c,x,y} G_{[k:k+V-1],c,u,v}
```
Optimization 1: Building Parallel Tasks

- Each input contributes to **spatially-grouped** outputs
- Reduce skippable computation per zero-checking
computes

contributes to

K ×

C ×
computes

contributes to
A vertical $S \times C$ input slice contributes to a horizontal $R \times K$ output slice.
Optimization 1: Building Parallel Tasks

- Check each element in input slice before moving to next output slice
Optimization 1: Building Parallel Tasks

- Check each element in input slice before moving to next output slice
- Tackle register spilling: shrink output slice to $R \times Q$
  - $Q$ is a factor of $K$
  - Goal: $R \times Q < \text{register budget}$
Optimization 1: Building Parallel Tasks

- Each parallel task executes a loop
Optimization 1: Building Parallel Tasks

- Each parallel task executes a loop
Optimization 1: Building Parallel Tasks

- Sweeps through the output width dimension $W$ and updates a $W \times Q$ output row
  - Called row sweep
Optimization 1: Building Parallel Tasks

- Each task computes $W \times Q$ distinct outputs: no race condition
- Creates many parallel tasks to avoid load imbalance
Optimization 2: Efficient Vector Register Usage

- Minimize memory accesses: keep outputs shared between iterations in registers
- Avoid register-to-register data transfer: cyclically rename registers
Optimization 2: Efficient Vector Register Usage

- Minimize memory accesses: keep outputs shared between iterations in registers
- Avoid register-to-register data transfer: cyclically rename registers

<table>
<thead>
<tr>
<th>Itr x</th>
<th>R0</th>
<th>R1</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Y_{x-2}$</td>
<td>$Y_{x-1}$</td>
<td>$Y_x$</td>
</tr>
</tbody>
</table>

Diagram: 3D representation of vector register usage.
Optimization 2: Efficient Vector Register Usage

- Minimize memory accesses: keep outputs shared between iterations in registers
- Avoid register-to-register data transfer: cyclically rename registers
Optimization 3: Reducing Branch Mispredictions

- Step 1: vectorize zero-checking along input channel dimension C

```plaintext
for i in 0 to N-1
    for c in 0 to C-1
        for y in S-1 to H+S-2
            for x in R-1 to W+R-2
                if D_{i,c,x,y} ≠ 0
                    ...
```

```plaintext
for i in 0 to N-1
    for c in 0 to C-V step V
        for y in S-1 to H+S-2
            for x in R-1 to W+R-2
                m_{[0:v-1]} = vect_cmp_neq_zero(D_{i,[c:c+V-1],x,y})
                for c' in 0 to V-1
                    if m_{c'} is true
                        ...
```

```
D_{i,[c:c+V-1],x,y}  
0 0 25 12 0 4 0 0  
```

```
m_{[0:v-1]}  
0 0 1 1 0 1 0 0  
```
Optimization 3: Reducing Branch Mispredictions

- Step 1: vectorize zero-checking along input channel dimension C
- Step 2: transform a series of if statements into a single loop

```
m[0:v-1]
0 0 1 1 0 1 0 0

... is true true true true true?

V if branches in total
```

```
for i in 0 to N-1
  for c in 0 to C-V step V
    for y in S-1 to H+S-2
      for x in R-1 to W+R-2
        m[0:v-1]=vect_cmp_neq_zero(D_{i,[c:c+V-1],x,y})
        for c' in 0 to V-1
          if m_{c'} is true
            ...
```
Optimization 3: Reducing Branch Mispredictions

- The transformation:

\[
\begin{align*}
&\text{m}[0:V-1] \quad \text{loop 3 times} \\
&\text{bit mask} \quad 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \\
&\text{input vector} \quad 0 \ 0 \ 25 \ 12 \ 0 \ 4 \ 0 \ 0
\end{align*}
\]
Optimization 3: Reducing Branch Mispredictions

- The transformation:

```

\begin{align*}
\text{bit mask} & \quad m_{[0:v-1]} \\
\text{input vector} & \quad D_{i,[c:c+V-1],x,y}
\end{align*}
```

- bit mask: 0 0 1 1 0 1 0 0 (iteration 1)
- input vector: 0 0 25 12 0 4 0 0

2 trailing zeros

pointer
Optimization 3: Reducing Branch Mispredictions

- The transformation:

\[
\begin{align*}
    m_{[0:v-1]} & \quad \text{iteration 1} \\
    \text{bit mask} & \quad / / / / 0 0 1 1 0 \\
    \text{input vector} & \quad 0 0 25 12 0 4 0 0
\end{align*}
\]

- Pointer: input vector is NOT shifted.
Optimization 3: Reducing Branch Mispredictions

- The transformation:

\[
\text{D}_{i, [c:c+v-1], x, y} \quad \text{m}_{[0:v-1]} \quad \text{iteration 2}
\]

- Bit mask:
  \[
  \begin{array}{cccccc}
  / & / & / & 0 & 0 & 1 & 1 & 0 \\
  \end{array}
  \]

- Input vector:
  \[
  \begin{array}{cccccc}
  0 & 0 & 25 & 12 & 0 & 4 & 0 & 0 \\
  \end{array}
  \]

+1 trailing zero

+1 pointer
Experimental Setup

- Implemented SparseTrain in *MKL-DNN*
- Compared SparseTrain with *MKL-DNN*’s:
  - Direct convolution (baseline)
  - im2col+GEMM
  - Winograd 3x3 convolution
  - Specialized 1x1 convolution
- Experimented on 6-core Skylake-X server CPU with AVX-512
- Evaluated training/inference of VGG and ResNet
ImageNet Performance

- SparseTrain outperforms other algorithms
- SparseTrain speedup:
  - Training: 1.3x-2.2x
  - Inference: 1.4x-1.9x
ImageNet Performance

- SparseTrain outperforms other algorithms
- SparseTrain speedup:
  - Training: 1.3x-2.2x
  - Inference: 1.4x-1.9x
Conclusion

- Presented SparseTrain:
  - First software-only algorithm to speedup DNN training by exploiting dynamic sparsity
  - Dynamically skips computation at runtime according to sparsity
  - Applicable to all three training phases
- Accelerates end-to-end training by 1.3x-2.2x and inference by 1.4x-1.9x
Thanks!

- Contact: Zhangxiaowen (Andy) Gong (gong15@Illinois.edu)
- Disclaimer: This presentation and recording belong to the authors. No distribution is allowed without the authors' permission.