Sparse Architectures

ISCA Tutorial (2019)
Website: http://eyeriss.mit.edu/tutorial.html

Joel Emer, Vivienne Sze, Yu-Hsin Chen
Motivation

- Leverage CNN sparsity to improve energy-efficiency
• The elements of each “rank” (dimension) are identified by their “coordinates”, e.g., rank K has coordinates 0, 1, 2

• Each element of the tensor is identified by the tuple of coordinates from each of its ranks, i.e., a “point”.
Tree-based Tensor Representation

N

K

0 1 2

K

0 1 2

N

0 1 2

Rank

Fiber
Tree-based Tensor Representation

Finding point (2, 1)
Each coordinate references a fiber
Rank-based Tensor Representation

Each coordinate references a fiber.
Rank-based Tensor Representation

Finding point (2, 1)
Rank-based Tensor Representation

What if tensor is sparse?
Sparse Tensor Representation
Sparse Tensor Representation

Finding point (2, 1)
Information in a Fiber

- Each fiber has set of (coordinate, “payload”) tuples

Payload: Reference to fiber in next rank

Payload: Value

Coordinate
Information in a Fiber

Method: \[ \text{payload} = \text{fiber.lookup}(\text{coordinate}) \]
Fiber Representation Choices

Each fiber has set of (coordinate, “payload”) tuples

- **Implicit Coordinates**
  - Uncompressed (no meta-data required)
  - Compressed – e.g., run length encoded

- **Explicit Coordinates**
  - E.g., coordinate list

- **Space efficiency of a representation depends on sparsity**
  - Compressed format can have overhead relative to uncompressed format for dense data
Compressed Implicit Coordinate Representations

• “Empty” coordinate compression via zero-run encoding
  – Run-length coding (RLE)
    • (run-length of zeros, non-zero payload)...
  – Significance map coding
    • (flag to indicate if non-zero, non-zero payload)...

• Payload encoding
  – Fixed length payload
  – Variable length payload
    • E.g., Huffman coding

• Efficiency of different traversal patterns through the tensor is affected by encoding, e.g., finding the payload for a particular coordinate...
Uncompressed/Compressed Representation

Rank

Fiber

N
K

N
K

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h

0 1 2
0 1 2

a c

0 1 2
0 1 2

g h
Uncompressed/Compressed Representation

Coordinate == Position

Position, Length

K: 0,2
N: 0,2

G: 2,0

H: 2,2

a: 0
b: 2
g: 0
h: 1
Uncompressed/Compressed Representation

Position, Length

Position, Length

Position
Uncompressed/Compressed Representation

Note: First element of pair is always sum of pair of elements in previous cell
Note: First element of pair is always the sum of pair of elements in previous cell, so length can be computed from next cell’s value.
Uncompressed/Compressed Representation

Position, Length

Extra cell for computing final length

Position

K

N

0 2 2 4

0 1 2

0 a c

2 0 3 1

2 3 3 h

N K 0 1 2 3 4
Compressed Sparse Row (CSR)
Index (non-zero position info – e.g., \textbf{IA} and \textbf{JA} for CSR) accounts for approximately half of storage for fine grained pruning

[Han et al., ICLR 2016]
Explicit Coordinate Representations

- Coordinate/Payload list
  - (coordinate, non-zero payload)...

- Hash table (per fiber)
  - (coordinate -> payload) mapping

- Hash table (per rank)
  - (fiber_id, coordinate -> payload) mapping

- Bit vector of non-zero coordinates
  - Uncompressed payload
Per Rank Tensor Representations

- Uncompressed [U]
  -
- Run-length Encoded [R]
  -
- Coordinate/Payload List [C]
  -
- Hash Table (per rank) \([H_r]\)
- Hash Table (per fiber) \([H_f]\)
  -

Inspired by collaboration with Kjolstad in [Kjolstad, TACO]
Payload Needed to Find Fiber in a Rank

- Uncompressed [U]
  - Payload*: None

- Run-length Encoded [R]
  - Payload*: Pointer to fiber data structure

- Coordinate/Payload List [C]
  - Payload*: Pointer to fiber data structure

- Hash Table (per rank) \([H_r]\)
  - Payload*: fiber_id

- Hash Table (per fiber) \([H_f]\)
  - Payload*: Pointer to fiber data structure

*Payload needed in preceding rank to perform “lookup():”
Notation for CSR

CSR: Tensor-UC/KN

Coordinate/Payload
Rank order
Uncompressed
Representation of Order of Ranks

Tensor-UC/KN -> CSR

Tensor-UC/NK -> CSC
Traversal Efficiency

Efficiency of different traversal patterns through the tensor is affected by encoding, e.g., finding the payload for a particular coordinate…

- **Operations:**
  - `payload = Tensor.Locate(coordinate… | point)`
  - `(coordinate, payload) = Tensor.Next(rank_traversal_order)`

Tensor.next() is a very common operation and its efficiency is highly dependent on representation, both order of ranks and representation of each rank….
Concordant traversal orders

CSR and CSC each has a natural (or “concordant”*) traversal order

* Term from Michael Pellauer
Example Traversal Efficiency

• Locate efficiency:
  – Uncompressed – direct reference - O(1)
  – Run length encoded – linear search – O(n)
  – Hash table – multiple references and compute – O(1)
  – Coordinate/Payload list – binary search – O(log n)

• Next efficiency (concordant traversal)
  – Uncompressed – sequential reference, good spatial locality - O(1)
  – Run length encoded – sequential reference – O(1)
  – Coordinate/Payload list - same as uncompressed

• Next efficiency (discordant traversal)
  – Essentially as good (or bad) as locate….
Merging Ranks

Tensor-CC/KN

Tensor-(C^2)/(KN)
Merging Ranks

• For efficiency one can form new representations where the data structure for two or more ranks are combined:

• Examples:
  – Tensor-\((C^2)\)
    List of (coordinate tuple, payload) - COO
  – Tensor-\((H^2)\)
    • Hash table with coordinate tuple as key
  – Tensor-\((U^2)\)
    • Flattened array
    • Coordinates can be recovered with modulo arithmetic on “position”
  – Tensor-\((R^2)\)
    • Flattened run-length encoded sequence
Fully-Connected (FC) Layer

filters

input fmaps

output fmaps

1

C

H

W

M

C

H

W
Output-Stationary Operations

Filter

M

K

N

0

1

2

0 1 2

a b c
d e f

0 1 2

a b c
d e f

0 1 2

a b c
d e f

0 1 2

a b c
d e f
Output-Stationary Lists

Filter

M

K

K

N

Filter

M

K

N
Output-Stationary Intersection Lists

Filter

2-2 is the only “effectual” computation
EIE: A Sparse Linear Algebra Engine

- Process Fully Connected Layers (after Deep Compression)
- Store weights column-wise in Run Length format (i.e., CSC format)
- Read relative column when input is non-zero

**Supports Fully Connected Layers Only**

\[ \begin{bmatrix} w_{0,0} & w_{0,1} & 0 & w_{0,3} \\ 0 & 0 & w_{1,2} & 0 \\ 0 & 0 & w_{2,1} & w_{2,3} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & w_{4,2} & w_{4,3} \\ w_{5,0} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{6,3} \\ 0 & w_{7,1} & 0 & 0 \end{bmatrix} \times \begin{bmatrix} a_1 \\ 0 \\ a_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \]

- Dequantize Weight
- Keep track of location

**Output Stationary Dataflow**

[Han et al., ISCA 2016]
PE Architecture
Impact of Representation on Dataflow

From SpMxV research

Compressed Sparse Row (CSR)  Compressed Sparse Column (CSC)

M N weights inputs outputs

weights inputs outputs

M N weights inputs outputs

Input stationary

Output stationary

CSC reduces memory bandwidth over CSR (when not $M \gg N$)

For DNN, $M = \#$ of filters,
$N = \#$ of weights per filter

[Dorrance et al., FPGA 2014]
Sparse Accelerators
1-D Output-Stationary Convolution

Weights * Inputs = Outputs

E = W - ceil(R/2)†

```
int i[W];       # Input activations
int w[R];       # Filter weights
int o[E];       # Output activations

for (e = 0; e < E; e++) {
  for (r = 0; r < R; r++) {
    o[e] += i[e+r]*w[r];
  }
}
```

What opportunity(ies) exist if some of the values are zero?

Can avoid reading operands, doing multiply and updating output

† Assuming: ‘valid’ style convolution
1-D Output-Stationary Convolution

\[ E = W - \text{ceil}(R/2) \]

**Weights**

\[
\begin{pmatrix}
8 & 0 & 6 \\
\end{pmatrix}
\]

**Inputs**

\[ W \]

**Outputs**

\[ E \]

```
int i[W];    # Input activations
int w[R];    # Filter weights
int o[E];    # Output activations

for (e = 0; e < E; e++) {
    for (r = 0; r < R; r++) {
        if (!w[r]) o[e] += i[e+r]*w[r];
    }
}
```

Saved energy but not time

\[ \uparrow \text{Assuming: ‘valid’ style convolution} \]
Skip **mult** and **mem reads** when image data is zero. Reduce **PE power** by 45%
Compressed Weights
Compressed Storage of Weights

Uncompressed Weights

\[
\text{Weights} \quad 8 \quad 0 \quad 6 \quad R
\]

\[
* \quad \text{Inputs} \quad W
\]

\[
= \quad \text{Outputs} \quad E = W - \text{ceil}(R/2)^\dagger
\]

Compressed Weights

\[
\text{Weights} \quad R
\]

\[
* \quad \text{Inputs} \quad W
\]

\[
= \quad \text{Outputs} \quad E = W - \text{ceil}(R/2)^\dagger
\]

\[\dagger\text{Assuming: ‘valid’ style convolution}\]
Compressed Storage of Weights

Uncompressed Weights

Weights

\[ \begin{array}{c}
8 \\
0 \\
6
\end{array} \]

* Inputs

\[ W \]

= Outputs

\[ E = W - \text{ceil}(R/2)^\dagger \]

Compressed Weights

Weights

\[ \begin{array}{c}
8 \\
6 \\
0 \\
2
\end{array} \]

* Inputs

\[ W \]

= Outputs

\[ E = W - \text{ceil}(R/2)^\dagger \]

\[ ^\dagger \text{Assuming: ‘valid’ style convolution} \]

Coordinate value in uncompressed storage
Compressed Weights 1-D Convolution

```
int i[W];         # Input activations
int wv[R], wc[R]; # Compressed filter weights
int o[E];        # Output activations

for (e = 0; e < E; e++) {
    for (r = 0; r < R; r++) {
        break if !wi[r].valid;
        o[e] += i[e+wc[r]] * wv[r];
    }
}
```

+ Weight values (wv)
+ Inputs
+ Outputs
+ Big enough to hold worst case size
+ Weights
+ R
+ Weight indices (wi)
+ Inputs * W = Outputs
+ E = W-ceil(R/2)†
+ 8  6
+ 0  2
+ Weight values (wv)
+ Weight indices (wi)
+ Table lookup of weight index
+ Direct access into compressed buffer
+ No

†: ceil(R/2)
To Extend to Other Dimensions of DNN

• Need to add loop nests for:
  – 2-D input activations and filters
  – Multiple input channels
  – Multiple output channels

• Add parallelism…
Compressed Inputs
Multi-Input Channel 1-D Convolution

Weights

Inputs

Outputs

\[ E = W - \text{ceil}(R/2) \]

\[
\begin{align*}
\text{int } i[C][W]; & \quad \# \text{ Multi-input channel activations} \\
\text{int } w[C][R]; & \quad \# \text{ Filter weights} \\
\text{int } o[E]; & \quad \# \text{ Output activations} \\
\end{align*}
\]

\[
\text{for } (w = 0; w < W; w++) \{ \\
\quad \text{for } (r = 0; r < R; r++) \{ \\
\quad\quad \text{parallel-for } (c = 0; c < C; c++) \{ \\
\quad\quad\quad o[w-r] += i[c][w] * w[c][r]; \\
\quad\quad\}\}
\quad\}
\}\]

Note opportunity for spatial sum

† Assuming: ‘valid’ style convolution
Multi-Input Channel 1-D Convolution

Should we compress along C or W dimension? Let’s see

† Assuming: ‘valid’ style convolution
Compressed Sparse W-dimension

int iv[C][W], icw[C][W];  // Compressed input activations
int w[C][R];            // Filter weights
int o[E];               // Output activations

for (w = 0; w < W; w++) {
    for (r = 0; r < R; r++) {
        parallel-for (c = 0; c < C; c++) {
            break if !icw[c][w].valid;
            o[icw[c][w]-r] += iv[c][w]*w[c][r];
        }
    }
}
Compressed Sparse C-dimension

```
int iv[C][W], icc[C][W];  // Compressed input activations
int w[C][R];             // Filter weights
int o[E];                // Output activations

for (r = 0; r < R; r++) {
    for (w = 0; w < W; w++) {
        parallel-for (c = 0; c < C; c++) {
            break if !icc[c][w].valid;
            o[w-r] += iv[c][w]*w[icc[c][w]][r];
        }
    }
}
```

Note we now have a synchronized spatial sum
Cnvlutin

- Process Convolution Layers
- Built on top of DaDianNao (4.49% area overhead)
- Speed up of 1.37x (1.52x with activation pruning)

[Albericio et al., ISCA 2016]
Compressing Inputs + Weights
Output Stationary – Sparse W&I

Weights * Inputs = Outputs

R  W   E = W-ceil(R/2)

| 8 | 0 | 6 | 4 | 0 | 0 | 0 | 3 | 0 | 0 | 8 | 0 | 2 | 1/3 | 0/3 | 1/3 | 0/3 |

int i[W];                # Input activations
int w[R];                # Filter weights
int o[E];                # Output activations

for (e = 0; e < E; e++) {
    parallel-for (r = 0; r < R; r++) {
        next if w[r] == 0;
        next if i[e+r] == 0;
        o[e] += i[e+r] * w[r];
    }
}

How often is work done in inner loop?  Not very much!
Flattened Inputs & Weights

```c
int i[C][W*H];        # Flattened input activations
int w[C][M*R*S];      # Flattened filter weights
int o[M][E][F];       # Output activations

for mrs2 = [0..MRS2) {
    for c2 = [0..C) {
        for wh1 = [0..WH1) {
            for mrs1 = [0..MRS1) {
                parallel-for wh0 = [0..WH0) x
                    mrs0 = [0..MRS0) {
                    m = Mcoord(mrs2, mrs1, mrs0);
                    e = Wcoord(wh1,wh0)-Rcoord(mrs2, mrs1, mrs0);
                    f = Hcoord(wh1,wh0)-Scoord(mrs2, mrs1, mrs0);
                    o[m][e][f] += i[c2][wh1*WH0+wh0]
                        * w[c2][mrs2*MRS1*MRS0+mrs1*MRS0+mrs0];
                }
            }
        }
    }
}
```

At inner loop inputs are stationary across steps mrs1

Any opportunity for spatial sum? No
Sparse CNN (SCNN)

- Architecture to exploit sparsity
Intuition behind SCNN

Forget the sliding windows based convolution

Observation
Each non-zero activation must be multiplied by each non-zero weight
Intuition behind SCNN

Forget the sliding windows based convolution

Observation
Each non-zero activation must be multiplied by each non-zero weight
Intuition behind SCNN

Forget the sliding windows based convolution

Observation
Each non-zero activation must be multiplied by each non-zero weight
Intuition behind SCNN

Forget the sliding windows based convolution

Observation
Each non-zero activation must be multiplied by each non-zero weight
Intuition behind SCNN

Forget the sliding windows based convolution

Observation
Each non-zero activation must be multiplied by each non-zero weight
Sparse CNN (SCNN)

**Supports Convolutional Layers**

- **Densely Packed Storage of Weights and Activations**
- **All-to all Multiplication of Weights and Activations**
- **Mechanism to Add to Scattered Partial Sums**

**Input Stationary Dataflow**

[Parashar et al., ISCA 2017]
SCNN PE microarchitecture

Sparse-compressed frontend

Flattened Weights
- \( wv[C][M*R*S] \)
- \( wim[C][M*R*S] \)
- \( wir[C][M*R*S] \)
- \( wis[C][M*R*S] \)

Flattened Input Activations
- \( iv[C][W*H] \)
- \( iiw[C][W*H] \)
- \( iih[C][W*H] \)

Weight FIFO (sparse)
- indices

IARAM (sparse)
- indices

Coordinate Computation
- \( m = Mcoord(mrs2, mrs1, mrs0); \)
- \( e = Wcoord(wh1,wh0) - Rcoord(mrs2, mrs1, mrs0); \)
- \( f = Hcoord(wh1,wh0) - Scoord(mrs2, mrs1, mrs0); \)

Dense backend

Flattened Weights

Output
- \( o[M][E][F]; \)

OARAM (sparse)
- indices

PPU: Halos ReLU Compress
- Neighbors

Buffer bank

A accumulator buffers
- \( F^* \)

Fxl multiplier array

[Parashar et al., ISCA 2017]
Flattened Inputs & Weights

```c
int iv[C][W*H] iiw[C][W*H], iih[C][W*H];
int wv[C][M*R*S], wim[C][M*R*S], wir[C][M*R*S], wis[C][M*R*S];
int o[M][E][F];

for mrs2 = [0..MRS2) {
    for c2 = [0..C) {
        for wh1 = [0..WH1) {
            for mrs1 = [0..MRS1) {
                parallel-for wh0 = [0..WH0) x
                mrs0 = [0..MRS0) {
                    break if !ii[c2][mrs2*MRS1*MRS0+mrs1*MRS0+mrs0].v;
                    break if !wi[c2][wh1*WH0+wh0].v;
                    m = Mcoord(mrs2, mrs1, mrs0);
                    e = Wcoord(wh1,wh0)-Rcoord(mrs2, mrs1, mrs0);
                    f = Hcoord(wh1,wh0)-Scoord(mrs2, mrs1, mrs0);
                    o[m][e][f] += i[c2][wh1*WH0+wh0]
                    * w[c2][mrs2*MRS1*MRS0+mrs1*MRS0+mrs0];
                }
            }
        }
    }
}
```
SCNN Energy Versus Density

[Parashar et al., ISCA 2017]
SCNN Latency Versus Density

[Parashar et al., ISCA 2017]
Eyeriss – V2

• Architecture to accommodate variety sparsity
Eyeriss v2: Balancing Flexibility and Efficiency

Efficiently supports

• Wide range of filter shapes
  – Large and Compact

• Different Layers
  – CONV, FC, depth wise, etc.

• Wide range of sparsity
  – Dense and Sparse

• Scalable architecture

Over an order of magnitude faster and more energy efficient than Eyeriss v1

[Chen et al., JETCAS 2019]
Eyeriss v2: Processing In PE

- $M_0$: # of output channels processed in a PE
- $C_0$: # of input channels processed in a PE
- $S$: filter width
- $U$: stride

Input Activations

Weights

Psums

Sliding Window $i$

Sliding Window $i+1$
Eyeriss v2: Compressed Data Format

Weight Matrix

CSC Compressed Data:
- data vector: \{a, b, c, d, e, f, g, h, i, j, k, l\}
- count vector: \{1, 0, 0, 0, 1, 2, 3, 1, 1, 0, 0, 0\}
- address vector: \{0, 2, 5, 6, 6, 7, 9, 9, 12\}
Eyeriss v2: PE Architecture

Input Activations

- \( iact \)
- \( \text{Addr} \)
- \( \text{SPad} \)
- \( 9 \times 4b \) Regs

Weights

- \( \text{Data} \)
- \( \text{SPad} \)
- \( 16 \times 12b \) Regs

- \( \text{Addr} \)
- \( \text{SPad} \)
- \( 16 \times 7b \) Regs

SRAM

- \( \text{Data} \)
- \( \text{SPad} \)
- \( 96 \times 24b \) SRAM

Psums Out

- \( \text{SPad} \)
- \( 32 \times 20b \) Regs

- \( \times 2 \) R/W ports

- \( \times 2 \) for SIMD

Psums In
Decision Tree in Eyeriss v2 PE

- **If the iact is zero**, the CSC format will ensure that it is not read from the spad and therefore no cycles are wasted.

- **If the iact is not zero**, its value will be fetched from the iact data SPad and passed to the next pipeline stage.
  - If there are non-zero weights corresponding to the non-zero iacts, they will be passed down the pipeline for computation. The zero weights will be skipped since the weights are also encoded with the CSC format.
  - If there are no non-zero weights corresponding to the non-zero iacts, the non-zero iacts will not be further passed down in the pipeline.
Summary

• Processing Irregular (Gather-Scatter)
  – If weights and inputs compressed to dense (gather); output scatter
  – If weights and inputs uncompressed sparse (scatter); output gather

• Overhead (must not exceed benefits of sparsity)
  – Storage of location information for compressed data
  – Logic for checking if either inputs are zero

• Underutilization
  – Number of parallel cores (tiling) → maximize parallelism, but minimize underutilization
  – Flatten to 1-D avoid fragmentation from limits of each dimension

• Workload Imbalance

• Lots of challenges in sparse deep neural network acceleration!