Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks

Yu-Hsin Chen¹, Joel Emer¹,², Vivienne Sze¹

¹ MIT   ² NVIDIA
Contributions of This Work

• A novel energy-efficient CNN dataflow that has been verified in a fabricated chip, *Eyeriss*.

• A taxonomy of CNN dataflows that classifies previous work into three categories.

• A framework that compares the energy efficiency of different dataflows under same area and CNN setup.

---

**Eyeriss** [ISSCC, 2016]

A reconfigurable CNN processor

35 fps @ 278 mW*

* AlexNet CONV layers
Deep Convolutional Neural Networks

Modern *deep* CNN: up to 1000 CONV layers

- CONV Layer
- Low-level Features
- CONV Layer
- High-level Features
Deep Convolutional Neural Networks

CONV Layer → Low-level Features → CONV Layer → High-level Features → FC Layers → Classes

1 – 3 layers
Deep Convolutional Neural Networks

Convolutions account for more than 90% of overall computation, dominating runtime and energy consumption.
High-Dimensional CNN Convolution

Input Image (Feature Map)  Output Image

Filter

Element-wise Multiplication

Partial Sum (psum) Accumulation

a pixel
High-Dimensional CNN Convolution

Input Image (Feature Map)  Output Image

Filter

Sliding Window Processing
High-Dimensional CNN Convolution

Many Input Channels (C)
High-Dimensional CNN Convolution

Many Filters (M)

Input Image

Output Image

Many Output Channels (M)
High-Dimensional CNN Convolution

Many Input Images (N) → Many Output Images (N)

Filters

\[ \text{Input} \rightarrow \text{Convolution} \rightarrow \text{Output} \]

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

\[ \text{C} \quad \text{C} \quad \text{C} \quad \text{C} \quad \text{C} \quad \text{C} \]

\[ \text{H} \quad \text{H} \quad \text{H} \quad \text{H} \quad \text{H} \quad \text{H} \]

\[ \text{N} \quad \text{N} \quad \text{N} \quad \text{N} \quad \text{N} \quad \text{N} \]

\[ \text{M} \quad \text{M} \quad \text{M} \quad \text{M} \quad \text{M} \quad \text{M} \]

\[ \text{E} \quad \text{E} \quad \text{E} \quad \text{E} \quad \text{E} \quad \text{E} \]

\[ \text{1} \quad \text{1} \quad \text{1} \quad \text{1} \quad \text{1} \quad \text{1} \]
Memory Access is the Bottleneck

**Memory Read**
- filter weight
- image pixel
- partial sum

**MAC**
- ALU

**Memory Write**
- updated partial sum

* multiply-and-accumulate
Memory Access is the Bottleneck

Memory Read | MAC* | Memory Write

DRAM | ALU | DRAM

* multiply-and-accumulate

Worst Case: all memory R/W are **DRAM** accesses

- Example: AlexNet [NIPS 2012] has 724M MACs
  → 2896M DRAM accesses required
Memory Access is the Bottleneck

Extra levels of local memory hierarchy
Memory Access is the Bottleneck

Extra levels of local memory hierarchy

Opportunities: data reuse
Types of Data Reuse in CNN

**Convolutional Reuse**
- CONV layers only (sliding window)

**Image Reuse**
- CONV and FC layers

**Filter Reuse**
- CONV and FC layers (batch size > 1)

**Reuse:**
- **Image pixels**
- **Filter weights**
Memory Access is the Bottleneck

Extra levels of local memory hierarchy

Opportunities: 1 data reuse

1 Can reduce DRAM reads of filter/image by up to $500 \times$**

** AlexNet CONV layers
Memory Access is the Bottleneck

1) Can reduce DRAM reads of filter/image by up to 500×

2) Partial sum accumulation does NOT have to access DRAM

Opportunities: 1 data reuse  2 local accumulation
Memory Access is the Bottleneck

Opportunities:

1. **data reuse**
   - Can reduce DRAM reads of filter/image by up to 500×

2. **local accumulation**
   - Partial sum accumulation does NOT have to access DRAM

   - Example: DRAM access in AlexNet can be reduced from 2896M to 61M (best case)
Spatial Architecture for CNN

**Local Memory Hierarchy**
- Global Buffer
- Direct inter-PE network
- PE-local memory (RF)

**Processing Element (PE)**
- Reg File: 0.5 – 1.0 kB
- Control
Low-Cost Local Data Access

fetch data to run a MAC here

Normalized Energy Cost*

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>ALU</th>
<th>ALU</th>
<th>ALU</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5 – 1.0 kB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NoC: 200 – 1000 PEs</td>
<td>PE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100 – 500 kB</td>
<td>Buffer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRAM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1× (Reference)

1×

2×

6×

200×

* measured from a commercial 65nm process
Low-Cost Local Data Access

How to exploit ① data reuse and ② local accumulation with *limited* low-cost local storage?

specialized processing dataflow required!

Normalized Energy Cost*

<table>
<thead>
<tr>
<th>DRAM</th>
<th>Buffer</th>
<th>RF</th>
<th>PE</th>
<th>ALU</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 – 500 kB</td>
<td>0.5 – 1.0 kB</td>
<td>200 – 1000 PEs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6×</td>
<td>2×</td>
<td>1×</td>
<td>1×</td>
<td></td>
</tr>
<tr>
<td>200×</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* measured from a commercial 65nm process
Taxonomy of Existing Dataflows

- Weight Stationary (WS)
- Output Stationary (OS)
- No Local Reuse (NLR)
Weight Stationary (WS)

- Minimize weight read energy consumption
  - maximize convolutional and filter reuse of weights

- Examples:
  - [Chakradhar, ISCA 2010]
  - [nn-X (NeuFlow), CVPRW 2014]
  - [Park, ISSCC 2015]
  - [Origami, GLSVLSI 2015]
Minimize partial sum R/W energy consumption
   - maximize local accumulation

Examples:

[Gupta, ICML 2015]  [ShiDianNao, ISCA 2015]
[Peemen, ICCD 2013]
No Local Reuse (NLR)

- Use a **large global buffer** as shared storage
  - Reduce **DRAM** access energy consumption

- **Examples:**
  - [DianNao, ASPLOS 2014]
  - [DaDianNao, MICRO 2014]
  - [Zhang, FPGA 2015]
Energy Efficiency Comparison

- Same total area
- AlexNet Configuration*
- 256 PEs
- Batch size = 16

Variants of OS

Normalized Energy/MAC

WS | OS_A | OS_B | OS_C | NLR

CNN Dataflows

Row Stationary

* AlexNet CONV layers
Energy-Efficient Dataflow: Row Stationary (RS)

- Maximize reuse and accumulation at RF
- Optimize for overall energy efficiency instead for only a certain data type
1D Row Convolution in PE

Filter \ast \text{Input Image} = \text{Output Image}
1D Row Convolution in PE

Filter

\[
\begin{array}{ccc}
  a & b & c \\
\end{array}
\]

Input Image

\[
\begin{array}{cccccc}
  a & b & c & d & e \\
\end{array}
\]

Partial Sums

\[
\begin{array}{ccc}
  a & b & c \\
\end{array}
\]
1D Row Convolution in PE

Filter

Input Image

Partial Sums

Reg File

PE
1D Row Convolution in PE

\[
\begin{array}{ccc}
\text{Filter} & \times & \text{Input Image} \\
\begin{array}{ccc}
a & b & c \\
\end{array} & \begin{array}{ccccc}
a & b & c & d & e \\
\end{array} & \begin{array}{ccc}
a & b & c \\
\end{array}
\end{array}
\]
1D Row Convolution in PE

Filter

\[
\begin{bmatrix}
a & b & c \\
\end{bmatrix}
\]

Input Image

\[
\begin{bmatrix}
a & b & c & d & e \\
\end{bmatrix}
\]

Partial Sums

\[
\begin{bmatrix}
a & b & c \\
\end{bmatrix}
\]
1D Row Convolution in PE

- Maximize row **convolutional reuse** in RF
  - Keep a **filter** row and **image** sliding window in RF
- Maximize row **psum accumulation** in RF
2D Convolution in PE Array

Row 1 * Row 1 = PE 1
2D Convolution in PE Array

Row 1

Row 2

Row 3

PE 1

PE 2

PE 3

=
2D Convolution in PE Array

Row 1 * Row 1 = Row 1
Row 2 * Row 2 = Row 2
Row 3 * Row 3 = Row 3
Row 1 * Row 2 = Row 2
Row 2 * Row 3 = Row 3
Row 3 * Row 4 = Row 4
2D Convolution in PE Array

Row 1

PE 1
Row 1 * Row 1

PE 2
Row 2 * Row 2

PE 3
Row 3 * Row 3

PE 4
Row 1 * Row 2

PE 5
Row 2 * Row 3

PE 6
Row 3 * Row 4

PE 7
Row 1 * Row 3

PE 8
Row 2 * Row 4

PE 9
Row 3 * Row 5

* * = 
* * = 
* * = 
Convolutional Reuse Maximized

Filter rows are reused across PEs horizontally
Convolutional Reuse Maximized

Image rows are reused across PEs diagonally.
Maximize 2D Accumulation in PE Array

Partial sums accumulate across PEs vertically
Dimensions Beyond 2D Convolution

1 Multiple Images  2 Multiple Filters  3 Multiple Channels
Filter Reuse in PE

1. Multiple Images
2. Multiple Filters
3. Multiple Channels

Processing in PE: concatenate image rows

Filter 1

Image 1 & 2

Psum 1 & 2

Channel 1

Row 1

Row 1

Row 1

Row 1
Image Reuse in PE

1 Multiple Images  2 Multiple Filters  3 Multiple Channels

Processing in PE: interleave filter rows

Channel 1

Filter 1

Row 1

* Image 1

Row 1

= Psum 1

Channel 1

Filter 2

Row 1

* Image 1

Row 1

= Psum 2

share the same image row

Filter 1 & 2

Image 1

Psum 1 & 2
Channel Accumulation in PE

1. Multiple Images
2. Multiple Filters
3. Multiple Channels

Filter 1
Image 1

Channel 1
Row 1 * Row 1 = Row 1

Channel 2
Row 1 * Row 1 = Row 1

accumulate psums

Row 1 + Row 1 = Row 1
Channel Accumulation in PE

1. Multiple Images
2. Multiple Filters
3. Multiple Channels

Processing in PE: interleave channels

accumulate psums
CNN Convolution – The Full Picture

Multiple images:
Filter 1 * Image 1 & 2 = Psum 1 & 2

Multiple filters:
Filter 1 & 2 * Image 1 = Psum 1 & 2

Multiple channels:
Filter 1 * Image 1 = Psum
Simulation Results

- Same total hardware area
- 256 PEs
- AlexNet Configuration
- Batch size = 16
Dataflow Comparison: CONV Layers

RS uses $1.4 \times - 2.5 \times$ lower energy than other dataflows
Dataflow Comparison: CONV Layers

RS optimizes for the best **overall** energy efficiency.
Dataflow Comparison: FC Layers

RS uses at least 1.3x lower energy than other dataflows
Row Stationary: Layer Breakdown

Total Energy

80% ↔ 20%

Normalized Energy
(1 MAC = 1)

CONV Layers
RF dominates

FC Layers
DRAM dominates
Summary

- We propose a **Row Stationary** (RS) dataflow to exploit the **low-cost local memories** in a spatial architecture.

- RS optimizes for best **overall** energy efficiency while existing CNN dataflows only focus on certain data types.

- RS has higher energy efficiency than existing dataflows
  - **1.4× – 2.5×** higher in **CONV** layers
  - at least **1.3×** higher in **FC** layers. (batch size ≥ 16)

- We have verified RS in a fabricated CNN processor chip, **Eyeriss**
Thank You

Learn more about Eyeriss at
http://eyeriss.mit.edu