Akida: A Low-Power Neuromorphic SoC for Event-Based Computation

Kristofor D. Carlson
Senior Research Scientist

This presentation discusses ongoing R&D work and the information in this presentation is subject to change.
Edge computing: compute where data is generated

Applications list:
- Smart home
- Industrial manufacturing
- Environment
- Medical/health care
- Transportation
- Energy management

Benefits:
- Latency reduction
- Power reduction
- Comm. reduction
- Increased security
- Increased privacy
Challenges: DNNs at the Edge

DNNs are a popular solution for:
- Speech recognition
- Image classification
- Anomaly detection
- Facial detection/recognition

DNNs are great at creating multi-level representations using automated feature extraction

Challenges:
- Training requirements
- Memory requirements
- Not built with event-based computing in mind
- Power requirements
- Computational requirements
- Adaptability
Design Goals

* Use a single **low-power hardware platform** to run:
  * Conventional DNN inference algorithms
  * Native SNNs and event-based algorithms
  * **On-chip** unsupervised learning algorithms
  * **Entire network** runs on Akida, host only passes data and receives results

* Additional goals:
  * Use well-known ML ecosystems for development
  * Implement a small set of useful computational operations
Akida System on a Chip

**CNN to SNN**
- Python Scripts
- GPUs

Efficient CNN Inference

**Native SNN**
- Python Scripts

Unsupervised, Rapid Learning of Repeating Patterns

---

**Spike Conversion Complex**

**Akida Neuron Fabric**

**Akida SoC**
- UART
- SPI
- JTAG
- USB3.0
- PCIe 2.1 (2x)
- PCIe 2.1 (2x)
- I²C
- I²S

** Akida USB stick**

**Akida PCIe board**

**FPGA PCIe board**

**Akida PCIe board**
Akida System on a Chip

CNN to SNN
- Python Scripts
- GPUs

Efficient CNN Inference

Akida System on a Chip

Akida SoC
- UART
- SPI
- JTAG
- USB3.0
- PCIe 2.1 (2x)
- I3C
- I2S

Spike Conversion Complex

Akida Neuron Fabric

PCle 2.1 (2x)

FPGA PCIe board
Akida PCIe board
Akida USB stick
Akida SoC
Low-Power CNN Inference

* **Applications**: Object classification, object detection, and speech recognition

* **Training**: Supervised with labelled dataset with activation and weight quantization

* **DNN algorithm focus**: Feed-forward CNNs

* **Purpose**: CNN inference on small, low-power devices

* **Solution**: Use event-based convolutions/separable convolutions

<table>
<thead>
<tr>
<th>Supported Layer Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolutional Input Layer</td>
</tr>
<tr>
<td>Generic Event-Encoding Layer</td>
</tr>
<tr>
<td>Standard Convolutional Layer</td>
</tr>
<tr>
<td>Separable Convolutional Layer</td>
</tr>
<tr>
<td>Fully-Connected Layer</td>
</tr>
<tr>
<td>Max Pooling Layer</td>
</tr>
<tr>
<td>Global Average Pooling Layer</td>
</tr>
</tbody>
</table>
Akida Execution Engine Running an Event-based CNN

CIFAR10 Classification by Akida

1/1 correct (100.00% acc.)

Out Neuron Activity

Plane  Car  Bird  Cat  Deer  Dog  Frog  Horse  Ship  Truck
CIFAR10 Data Set Results

- **Training Images:**
  - 60,000 Color Images
  - 32 X 32 Resolution
  - 10 Classes

- **Testing Images:**
  - 10,000 Images

- **Network:**
  - VGG-like
  - Ternary weights
  - Binary activations

- **Results:**
  - Accuracy: **Top-1:** ~92%

<table>
<thead>
<tr>
<th>Layer #</th>
<th>Type/Stride</th>
<th>Filter Shape</th>
<th>Input Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Conv/s1</td>
<td>3x3x3x128</td>
<td>32x32x3</td>
</tr>
<tr>
<td>2</td>
<td>Conv/s1 MP</td>
<td>3x3x128x128</td>
<td>32x32x128</td>
</tr>
<tr>
<td>3</td>
<td>Conv/s1</td>
<td>3x3x128x256</td>
<td>16x16x128</td>
</tr>
<tr>
<td>4</td>
<td>Conv/s1 MP</td>
<td>3x3x256x256</td>
<td>16x16x256</td>
</tr>
<tr>
<td>5</td>
<td>Conv/s1</td>
<td>3x3x256x512</td>
<td>8x8x256</td>
</tr>
<tr>
<td>6</td>
<td>Conv/s1 MP</td>
<td>3x3x512x512</td>
<td>8x8x512</td>
</tr>
<tr>
<td>7</td>
<td>Fully Conn</td>
<td>8192x1024</td>
<td>4x4x512</td>
</tr>
<tr>
<td>8</td>
<td>Fully Conn</td>
<td>1024x1024</td>
<td>1024</td>
</tr>
<tr>
<td>9</td>
<td>Fully Conn</td>
<td>1024x10</td>
<td>10</td>
</tr>
</tbody>
</table>

Total Parameters: 14,018,560

Equivalent Synapses: 655,239,168  — Benefit of using convolution
ImageNet Data Set Results

*Training Images:
  * ~1.2M Color Images
  * 224 X 224 Resolution
  * 1,000 Classes

*Testing Images:
  * ~50,000 Images

*Network:
  * Adapted MobileNet V1*
  * Akida-Compatible:
    * 4-bit weights/activations
    * Single 2-bit weights layer

*Results:
  * Our Accuracy: **Top-1: ~65.6%**

---

## Akida SoC
Low-Power CNN Inference: Convolutional Ops

<table>
<thead>
<tr>
<th>Supported</th>
<th>Kernel Size</th>
<th>Stride</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image convolution (RGB888 or grayscale)</td>
<td>Conv: 3x3, 5x5, 7x7 Max Pool: 2x2</td>
<td>1,2,3</td>
<td>Same, valid</td>
</tr>
<tr>
<td>Standard convolution</td>
<td>1x1, 3x3, 5x5, 7x7</td>
<td>1</td>
<td>Same</td>
</tr>
<tr>
<td>Point wise convolution</td>
<td>1x1</td>
<td>1</td>
<td>Same</td>
</tr>
<tr>
<td>Depth-wise convolution</td>
<td>3x3, 5x5, 7x7</td>
<td>1</td>
<td>Same</td>
</tr>
<tr>
<td>Max pooling</td>
<td>2x2, 3x3</td>
<td>1, 2, 3 (only for 3x3)</td>
<td>N/A</td>
</tr>
<tr>
<td>Global average pooling</td>
<td>$W_{Input} \times H_{Input}$</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Weight (signed) & Activation

<table>
<thead>
<tr>
<th>Weight (signed)</th>
<th>Activation</th>
<th>No MACs</th>
<th>MACs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – 2-bit</td>
<td>1 – 2-bit</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>&gt;2-bit</td>
<td>1 – 2-bit</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>1 – 2-bit</td>
<td>&gt;2-bit</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>4-bit</td>
<td>4-bit</td>
<td>✗</td>
<td>✔️</td>
</tr>
</tbody>
</table>

**Lower power** | **Higher Power**
How Can We Compute Convolutions Efficiently?

- Reduce the number of required operations
  - Event-based convolutional and fully-connected layers
  - Separable convolutions yield fewer computations*

- Reduce cost of each operation
  - Especially important for SNNs (cost of each event matters!)

- Reduce memory requirements
  - Separable convolutions give you fewer parameters
  - Low-precision weights (1-4 bit)
Akida System on a Chip

**CNN to SNN**
- Python Scripts
- GPUs
  - Efficient CNN Inference

**Native SNN**
- Python Scripts
  - Unsupervised, Rapid Learning of Repeating Patterns

---

**Akida SoC**
- UART
- SPI
- JTAG
- USB3.0
- PCIe 2.1 (2x)
- I3C
- I2S

**Spike Conversion Complex**
**Akida Neuron Fabric**
**PCIe 2.1 (2x)**

**Interfaces**
- PCIe 2.1 (2x)
- DDR4
- UART
- SPI
- JTAG
- USB3.0
- I3C
- I2S

**Platforms**
- FPGA PCIe board
- Akida PCIe board
- Akida USB stick
Native SNN

Python Scripts

Unsupervised, Rapid Learning of Repeating Patterns

Akida System on a Chip

Akida SoC

UART

SPI

JTAG

USB3.0

PCle 2.1 (2x)

I3C

I2S

Spike Conversion Complex

Akida Neuron Fabric

FPGA PCIe board

Akida PCIe board

Akida USB stick
Akida SoC
Low-Power Unsupervised Learning for Pattern Recognition

• Proprietary learning rule:
  • Is an STDP-inspired learning rule

• Includes:
  • Homeostatic mechanisms
  • Competition mechanisms
  • Extremely simple and efficient implementation

• Excels at finding and learning input patterns embedded in noise with very few repetitions (on average 3–5)
Akida SoC
On-Chip Learning Application Using DVS Camera

* Akida trained on hand-gestures using an event-based camera
* Pre-trained on 3 hand-gestures
* Learns class 5 in real-time
* Uses our proprietary unsupervised learning rule
Akida SoC
On-Chip Learning Cybersecurity Application

- Akida incorporates data-to-spike converters to process tabularized data from the PCAP file.
- Trained on the CIC-IDS-2017 dataset, 2.5M lines corresponding to captured internet traffic with 14 categories of attack, using native on-chip learning.
- 97.4% accuracy F1 score = 0.97 using 10% of never-seen samples as test set.
- Training time: ~20 minutes/epoch (used 1 epoch).
Akida Development Environment

Akida Development Environment (ADE)

Python Scripts
- Data Pre-Processing
- Specify: Data Model Training Mode
- Instrumentation/Visualization

Data

Akida Model Zoo
- MNIST
- CIFAR-10
- DVS

Akida Execution Engine (AEE)
- Data->Spike Converter
- Akida Neuron Model
- Akida Training Methods

Supporting Tools
- Jupyter
- Python
- NumPy
- Matplotlib

Compiled C/C++ executable

Scripts or Libraries
Akida Development Environment

CNN2SNN Toolbox

**CNN2SNN training - quantization**
- Python Scripts
- Keras
- TensorFlow

- Use existing deep learning framework (Keras + TensorFlow)
- Need to follow specs defining Akida compatible CNN components

**Conversion to Akida SNN**
- Python Scripts
- Akida SNN

- .yml file (network)
- Binary file (weights)

Supporting Tools
- Python
- NumPy
- matplotlib
Thank you

For more details contact:

Peter AJ van der Made, CTO: pmade@brainchip.com
Advantages and Disadvantages

- The classification speed is highly dependent on the host computer.
- Benchmarks of frames per second have little relevance to the MAC accelerator speed itself.
- Power figures are also highly dependent on the host computer, which runs all of the neural network functions.
- No autonomous learning functions, limited to CNN only.
- This architecture is highly suited to run any kind of dedicated Artificial Neural Network trained with Deep Learning.
Advantages and Disadvantages

• The entire network runs on the chip, the host only needs to pass preprocessed data to the network, and processed metadata is returned (classification etc).

• This frees up the host memory and cycles for other tasks.

• Power and speed benchmarks are clearly defined for the Akida chip.

• On-chip learning, optional convolutional, pooling and fully connected layers.

• Limited to running native learning SNN, autonomous learning CNN, and feed-forward CNN trained with DL.
Advantages and Disadvantages

• The entire event-based CNN network runs on the chip, the network configuration is contained in non-volatile memory, no host is needed

• The on-chip CPU can be used for pre-processing of data

• Power and speed benchmarks are clearly defined for the Akida chip

• On-chip learning, optional convolutional, pooling and fully connected layers

• Limited to running native learning SNN, autonomous learning CNN, and feed-forward CNN trained with DL.
Advantages and Disadvantages

• The entire network runs on the chip, no host is needed. The chip is initialized by the on-chip ARM processor, Meta-data output

• The on-chip ARM processor can be used for pre-processing of data

• Power and speed benchmarks are clearly defined for the Akida chip

• Network configuration, layers, connections are stored in non-volatile memory

• On-chip autonomous learning