Analytical Modeling of Sparse Tensor Accelerators

Nellie Wu

PIs: Joel Emer, Vivienne Sze

May 2022
Tensor Algebra is Widely Used in Many Applications

- Deep Learning
- Graph Analytics
- Circuit Simulations

General-purpose Processors

Data and Computation Intensive

× Slow and/or Energy Consuming
Domain-Specific Accelerators

Example Accelerator Architecture

- Efficient data movement and compute
- Various underlying technologies, e.g., digital or analog

Achieve high energy efficiency and processing speed
Sampled Tensor Algebra Accelerator Designs

Digital-Compute Accelerator Designs

Each accelerator has its own: (1) target applications (2) architecture properties (3) processing schedule

Form a large and diverse design space

Works we contributed to

Analog-Compute Accelerator Designs

CASCADe [MICRO2019]

ISAAC [ISCA2016]

DrAcc [DAC2018]
Many Applications Use Sparse Tensors

[Hegde, et.al., MICRO 2019]
Exploiting Sparsity

Sparse data can be compressed

\[ \text{anything} \times 0 = 0 \]

\[ \text{anything} + 0 = \text{anything} \]

Can save space and energy

Can save time and energy
Larger and More Diverse Design Space

Digital-Compute Accelerator Designs

- Eyeriss [JSSC2017]
- Simba [MICRO2019]
- ExTensor [MICRO2019]
- Eyeriss V2 [JETCAS2019]
- SCNN [ISCA2017]

Analog-Compute Accelerator Designs

- CASCADE [MICRO2019]
- ISAAC [ISCA2016]
- DrAcc [DAC2018]
- SpaceA [HPCA2021]
- SNrram [DAC2018]
- Sparse-ReRAM [ISCA2019]

Works we contributed to:
- Eyeriss [JSSC2017]
- SCNN [ISCA2017]
- DianNao [ASPLOS2014]
Important to perform apple-to-apple comparison of existing designs and fast exploration of new potential designs

A fast and generally applicable modeling framework is necessary
**Existing Work**

### Detailed Design-Specific Models

- **e.g., Eyeriss V2 [JETCAS 2019], Simba [MICRO 2019]**

- **Capture design details**
- **Accurate**
- **Slow modeling speed**
- **Can be parameterized, but very limited modeling flexibility**

### Flexible Dense Analytical Models

- **e.g., Timeloop [ISPASS 2019], CoSA [ISCA 2021]**

- **High-level design characteristics**
- **Reasonably accurate**
- **Fast modeling speed**
- **Flexible for dense designs but do not capture sparse designs**

---

---
Sparseloop: Analytical Modeling for Tensor Accelerators

Workload

Mapping

Architecture

Sparse Acceleration Features

Energy

Cycles
Classification of Important Design Aspects

- **Dataflow**
  Processing schedule that defines uncompressed data movement and dense compute in time and space

- **Sparse Accelerations**
  Processing schedule that defines how to improve hardware performance when workloads are sparse

- **Microarchitecture**
  Detailed architecture properties that define the exact cost of different hardware operations, e.g., storage access, compute
Modularized Modeling Process

**Sparseloop**

- **Step 1: Dataflow Modeling**
- **Step 2: Sparse Modeling**
- **Step 3: Micro-Architectural Modeling**

**Architectures**
- Global Buffer (GLB)
- PE0
- PE1
- PE2
- PE3

**Sparse Acceleration Features**
- format
- gating
- skipping

**Workload**

**Mapping**

*→ a mapping is an instance of a dataflow*
Dataflow Modeling

Analyzes Uncompressed Data Movement and Dense Compute
Answer dataflow related questions

- Which tensor is stationary for how long at each storage level?
- How much data is transferred between storages?
- How many compute happened?
- ...

fast analysis as most tiles are identical

Derives Uncompressed Data Movement and Dense Compute
What if this is a Sparse Design?

Hardware Sparse Acceleration Features (SAFs)

Format:
Choose tensor representations to save necessary storage spaces and energy associated zero accesses

Gating:
Explicitly eliminate ineffectual storage accesses and computes by letting the hardware unit staying idle for the cycle to save energy

Skipping:
Explicitly eliminate ineffectual storage accesses and computes by skipping the cycle to save energy and time
Each Feature Can Be Implemented Differently

Hardware Sparse Acceleration Features (SAFs)

Format:
Choose tensor representations to save necessary storage spaces and energy associated zero accesses

Gating:
Explicitly eliminate ineffectual storage accesses and computes by letting the hardware unit staying idle for the cycle to save energy

Skipping:
Explicitly eliminate ineffectual storage accesses and computes by skipping the cycle to save energy and time

What is the chosen format?
Do all tensors share the same format?
When is a storage access gated?
How much is the compute able to skip ahead?
At which storage level is the skipping performed?
What is the criteria for skipping?
Example Gating Implementation

Example Workload: Dot Product of Vectors

\[ Z = \sum_{k} A_k \times B_k \]

Accelerator Architecture

Buffer

Multiply-Accumulate Unit

Mapping

\[ \text{for } k \text{ in } [0:K) \]
\[ Z += A[k] \times B[k] \]

Gate buffer read of B vector and compute when A value is zero
Example Gating Implementation

Example Workload:
Dot Product of Vectors

\[ Z = \sum_{k} A_k \cdot B_k \]

\[
\begin{align*}
0 & \quad 0 \\
0 & \quad h \\
c & \quad i \\
d & \quad j \\
0 & \quad 0 \\
f & \quad l \\
\end{align*}
\]

\[
\begin{align*}
K & \quad \cdot \\
A & \quad B \\
Z & \quad \end{align*}
\]

\[ c_i + d_j + f_l \]

\[
\begin{align*}
\text{for } k \text{ in } [0:K) \\
Z & \text{+= } A[k] \cdot B[k]
\end{align*}
\]

Gate buffer read of B vector and compute when A value is zero
**Example Gating Implementation**

**Example Workload:**
Dot Product of Vectors

\[
Z = \sum_{k} A_k \times B_k
\]

**Accelerator Architecture**

- **Buffer**
  - 0 0 c d 0 f
  - 0 h i j 0 l

- **Multiply-Accumulate Unit**
  - 0

*Z data movements not shown

**Mapping**

\[
\text{for } k \text{ in } [0:K) \\
Z += A[k] \times B[k]
\]

Gate buffer read of B vector and compute when A value is zero
Example Gating Implementation

Example Workload: Dot Product of Vectors

\[ Z = \sum_{k} A_k \times B_k \]

\[
\begin{align*}
K & \quad \cdot \\
A & \quad 0 & c & d & 0 & f \\
B & \quad 0 & h & i & j & 0 & l \\
Z & \quad c_i + d_j + f_l \\
\end{align*}
\]

for \( k \) in \([0:K)\)

\[ Z += A[k] \times B[k] \]

Accelerator Architecture

Buffer

Multiply-Accumulate Unit

Gate buffer read of B vector and compute when A value is zero

*Z data movements not shown
SAF’s Impact Is Data-Dependent

We would like to have reasonably accurate AND fast data-dependent modeling

Existing Approaches

- Design-Specific Models based on exact data \( \times \) Too Slow
- Flexible Dense Models: based on abstract tiles shapes \( \times \) Inaccurate
Statistical Density Analysis

Buffer Multiplier

Main Memory

A B Z

Tile Density Probability Distribution

Probability

0 0.1 ... 1.0

Tile Density

dataflow sparse μarch
Example Density Models

Fixed-Structured
Same fixed density in all tiles

Uniform Distribution
Randomly distributed nonzeros

Banded Distribution
Diagonally distributed nonzeros

Fixed density

Larger density deviation

Smaller density deviation

Much larger density

Much smaller density
Sparse Modeling Can Be Complex

Answer Sparse Modeling Questions

- How much reads can be gated/skipped?
- What is the savings and overhead of compressed format?
- What’s the implication of a gating/skipping on lower levels?
- ...

*Z data movements not shown
Modularized And Extensible Analysis

Density Characterization Module
Format Agnostic Statistical Characterization*

Gating/Skipping Analysis Module

Data Representation Analysis Module

Format Models

*more details in [1]

Appropriately Postprocess Dense Traffic

Density Characterization Module
Format Agnostic Statistical Characterization*

Gating/Skipping Analysis Module

Data Representation Analysis Module

Scaling Module

% savings on read/write/compute

Metadata overhead
Data tile occupancy

Dense Traffic
Sparse Traffic

μarch

*more details in [1]

Sparse Traffic

Main Memory
- # actual reads
- # skipped reads
- # gated reads
- # actual metadata reads
- # actual writes
- # skipped writes
- # gated writes

Buffer[0..255]
...
Multiplier[0..255]
  - # actual multiply
  - # skipped multiply
  - # gated multiply

Micro-architectural properties
- Block size of SRAM
- Multiplier resolution
- ...

Energy Efficiency Cycles
Modularized Modeling Process

**Step 1: Dataflow Modeling**

**Step 2: Sparse Modeling**

**Step 3: Micro-Architectural Modeling**

**Sparse Acceleration Features**

- format
- gating
- skipping

**Architecture**

- Global Buffer (GLB)
- PE0
- PE1
- PE2
- PE3

**Mapping**

```
for (int x = 0; x < W; x++) {
    for (int y = 0; y < H; y++) {
        if (x == 0 || y == 0 || x == W-1 || y == H-1) { // boundary conditions
            // code here
        }
    }
}
```

**Sparseloop**

- Dense traffic stats
- Sparse traffic stats

**Workload**

**Energy**

**Cycles**

*a mapping is an instance of a dataflow*
Experimental Results

Speed, Accuracy, and Flexibility
Fast Simulation Speed

Evaluation metric: Computes Simulated Per Host Cycle (CPHC)

The machine that runs modeling framework, e.g., CPU

*Cycle-level Simulators, e.g., STONNE, often have <0.5 CHPC*
Fast Simulation Speed

Evaluation metric: Computes Simulated Per Host Cycle (CPHC)

The machine that runs modeling framework, e.g., CPU

*Cycle-level Simulators, e.g., STONNE, often have <0.5 CPHC*

### Sparseloop’s CPHCs

<table>
<thead>
<tr>
<th>Accelerator Designs</th>
<th>Workloads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ResNet50</td>
</tr>
<tr>
<td>Eyeriss</td>
<td>5.2k</td>
</tr>
<tr>
<td>Eyeriss V2 PE</td>
<td>2.7k</td>
</tr>
<tr>
<td>SCNN</td>
<td>1.1k</td>
</tr>
</tbody>
</table>

Sparseloop has a CPHC of >1.1k, >2000x faster than STONNE
Accurate Modeling

Less than 1% error comparing to results generated by a custom analytical SCNN simulator with statistical modeling
Accurate Modeling

Validation on Eyeriss V2 PE [JETCAS19] Architecture

Less than 7% error comparing to results generated by a custom Eyeriss V2 analytical simulator with actual data modeling
Rapid Exploration of Various Designs

Various workloads

Various combinations of SAF implementations at different architecture levels
Rapid Exploration of Various Designs

Various combinations of SAF implementations at different architecture levels

Sparseloop helps to quickly identify the savings and overheads for the diverse implementations
Summary

• **Sparseloop** is an analytical modeling framework that
  
  – Enables tensor accelerator design space exploration
  
  – Decouples the complex modeling process into three more tractable steps
    
    • Dataflow modeling
    
    • Sparse acceleration features modeling
    
    • Microarchitectural modeling
  
  – Uses statistical analysis to perform fast and reasonably accurate data-dependent modeling for sparse acceleration features

• Sparseloop is >2000x faster than cycle-level simulators

• We validate Sparseloop on well-known accelerator designs

• We show that Sparseloop helps designers to understand the critical factors involved in sparse tensor architecture evaluation and design
Resources

• Paper

• ISCA Tutorial
  – http://accelergy.mit.edu/sparse_tutorial.html
  – Open-sourced tools and infrastructure docker
  – Slide decks
  – Videos