Using a CGRA with Dynamic Partial Reconfiguration

Taeyoung Kong
Kalhan Koul, Priyanka Raina, Mark Horowitz, and Christopher Torng
Stanford University
Motivation - Multi-Task Workloads

=> Exploit *Dynamic Partial Reconfiguration* of CGRA to adapt to such workloads
Fast DPR allows a rapid change of underlying kernel

Flexible resource allocation of MEM units and PE units increases resource utilization
Compiler prepares several versions of the single kernel (Variants)
- different hardware resources
- different runtime
Scheduler selects one of the kernel variants at runtime based on
- remaining runtime requirements
- resource availability
Simulator - Python-Based Performance Simulator

Performance simulator allows easy exploration of CGRA and scheduler

Stanford University
Result - Image Processing + ML Workload

Lane detection + Object detection

1.6x speedup with
- DPR & flexible resource allocation
- resource-aware scheduler
Thank You
Scorch
A library for sparse machine learning

Bobby Yan*, Alexander J. Root*, Trevor Gale*†, Fredrik Kjolstad*
*Stanford University, †Google Research
Sparsity comes from data and model design

- Mixture of experts
- Graph neural networks
- Sparse transformers
- Recommender systems
Programming model landscape is fragmented

- torch.sparse
- tf.sparse
- jax.sparse
- PyG
- DGL
- TVM
- MLIR Sparse
- cuSPARSE
- cuSPARSE
- MKL-DNN
Programming model landscape can be unified

torch.sparse  tf.sparse  jax.sparse  PyG  DGL

tvm  MLIR Sparse  cuSPARSE  cuSPARSE  MKL-DNN

Scorch
Compiling Declarative Recurrences To Imperative Code

Shiv Sundram, Muhammad Usman Tariq, Fred Kjolstad

August 2023
Three classes of algorithms

1. Linear Solvers
   (LU, Cholesky)
Three classes of algorithms

1. **Linear Solvers**
   (LU, Cholesky)

2. **Dynamic Programs**
   (Sequence Alignment, Bellman)

3. 

Stanford University
Three classes of algorithms

1. **Linear Solvers**  
   (LU, Cholesky)

2. **Dynamic Programs**  
   (Sequence Alignment, Bellman)

3. **Graph Algorithms**  
   (Floyd-Warshall, Viterbi)
All expressible in a common language

Recurrence Equations

\[ L_{ij} = \frac{(A_{ij} - \sum_{k=0}^{j} L_{ik}L_{jk})}{L_{jj}} \]

\[ \text{if} \ j < i \]

\[ L_{ij} = \sqrt{A_{ij} - \sum_{k=0}^{j} L_{jk}L_{jk}} \]

\[ \text{if} \ j = i \]
All expressible in a common language

<table>
<thead>
<tr>
<th>Recurrence Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_{ij} = (A_{ij} - \sum_{k=0}^{j} L_{ik}L_{jk})/L_{jj}$</td>
</tr>
<tr>
<td>$: j &lt; i$</td>
</tr>
<tr>
<td>$L_{ij} = \sqrt{A_{ij} - \sum_{k=0}^{j} L_{jk}L_{jk}}$</td>
</tr>
<tr>
<td>$: j = i$</td>
</tr>
</tbody>
</table>
All expressible in a common language

**Recurrence Equations**

\[ L_{ij} = \left( A_{ij} - \sum_{k=0}^{j} L_{ik} L_{jk} \right) / L_{jj} \]

: \( j < i \)

\[ L_{ij} = \sqrt{A_{ij} - \sum_{k=0}^{j} L_{jk} L_{jk}} \]

: \( j = i \)

**Schedule**

- loop ordering
- parallelization

**Storage**

- Dense (row/column major)
- Sparse (CSR)
- Sparse (CSC)
All expressible in a common language

**Recurrence Equations**

\[ L_{ij} = \frac{(A_{ij} - \sum_{k=0}^{j} L_{ik}L_{jk})}{L_{jj}} \]

: \( j < i \)

\[ L_{ij} = \sqrt{A_{ij} - \sum_{k=0}^{j} L_{jk}L_{jk}} \]

: \( j = i \)

**Schedule**

- loop ordering
- parallelization

**Storage**

- Dense (row/column major)
- Sparse (CSR)
- Sparse (CSC)

**Dependency Analysis**

**Lower to IR Recurrence Index Notation**

for \( i<N \):

for \( j<i \):

for all \( k<j \):

\[ L_{1ij} = L_{ij}L_{ik} \]

\[ L_{ij} = (A_{ij} - L_{1ij})/L_{jj} \]

\[ L_{1ii} = L_{ij}L_{ij} \]

\[ L_{ii} = \sqrt{A_{ii} - L_{1ii}} \]
DSL for Recurrences

+Emit parallel C code
+Sparsity
+Loop ordering
  (nontrivial for recurrences)
+Loop-fusion
DSL for Recurrences

+Emit parallel C code
+Sparsity
+Loop ordering (nontrivial for recurrences)
+Loop-fusion

- Viterbi Equation (info. theory)
- Floyd-Warshall (shortest paths)
- Needleman-Wunsch (seq. alignment)
- Cholesky Decomposition (direct solver)
- Gauss-Seidel (iterative solver)
- Tensor Algebra (tensor algebra)

Performance improvements or parity with existing libraries
Compiling Sparse Machine Learning to Streaming Dataflow

Rubens Lacouture, Olivia Hsu, Nathan Zhang, Ritvik Sharma, Kunle Olukotun, Fred Kjolstad
Expressing ML Applications in SAM

• Prior work, Sparse Abstract Machine (SAM), is not enough to express sparse ML

We explored these state-of-the-art models:

<table>
<thead>
<tr>
<th>Transformers</th>
<th>GNNs</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-Tensor multiplication ✓</td>
<td>SpMM ✓</td>
</tr>
<tr>
<td>4-Tensor multiplication ✓</td>
<td>SDDMM ✓</td>
</tr>
<tr>
<td>Tensor addition ✓</td>
<td>SpMV ✓</td>
</tr>
<tr>
<td>Slicing ✗</td>
<td>Slicing ✗</td>
</tr>
<tr>
<td>Reshape/transpose/split(concat) ✗</td>
<td>Reshape/transpose ✗</td>
</tr>
<tr>
<td>Array programming ✗</td>
<td></td>
</tr>
<tr>
<td>Data/mask generator ✗</td>
<td>Data/mask generator ✗</td>
</tr>
</tbody>
</table>

✓ Supported in SAM
✗ Not in SAM
Expressing ML Applications in SAM

- Augment SAM to support the large space of sparse ML
- New hardware primitives
- DL framework, new compiler optimizations
- Faster concurrent simulation platform
- Leverage its tensor algebra compilation capabilities

**Filter**
\[ f(i, j) = i > j \]
Triangular lower

**Unary ALU**
\[ f(x) = \exp(x) \]
Exp

```
3,2,1
3,2,1
3,2,1
```
Expressing ML Applications in SAM

So now it can

Attn=softmax(αQ*Kᵀ)*V

We explored these state-of-the-art models:

- Transformers
- GNNs

- 3-Tensor multiplication ✅
- 4-Tensor multiplication ❌
- Tensor addition ✅
- Slicing ✅
- Reshape/transpose/split/concat ✅
- Array programming ✅
- Data/mask generator ✅
- SpMM ✅
- SDDMM ✓
- SpMV ✓
- Slicing ✓
- Reshape/transpose ✓
- Data/mask generator ✓

Not in SAM

Supported in SAM extended

PyTorch

TorchScript
Parse

Optimize

PyTorch IR

Lower

Optimize

DL Compiler

SAM Extended

Simulator

Sparse ML Dataflow Accelerator
Reuse and Traffic with Tiling for Sparse Accelerators

A sparse accelerator’s traffic vs it’s optimal memory traffic (red dots)
Reuse and Traffic with Tiling for Sparse Accelerators

Different Tiling Strategies

Static Coord Space

Static Position Space

Dynamic Coord Space
Reuse and Traffic with Tiling for Sparse Accelerators

• Extending our sparse framework to support different tiling strategies
• Optimize tiling for improved memory reuse & traffic based on collected statistics for suitesparse matrices

SAM primitives and graphs for Tiling

Suitesparse matrices data distribution
Sparse Shape Operators

Alexander J. Root, Bobby Yan, Peiming Liu, Aart Bik, Fredrik Kjolstad
Array programming is ubiquitous
Sparse arrays are used across many domains
Array shapes are often manipulated

\[ b = \begin{bmatrix} X & 0 & 0 \\ 0 & Y & 0 \\ 0 & 0 & Z \end{bmatrix} a \]
Shape operators

- `reshape(16)`
- `reshape(4, 2, 2)`
- `concat(1)`
- `slice(1:4, 2)`
Sparse shape operator support is sparse

- Limited data structure support
- Lacking fusion across function calls
Sparse shape operator support is sparse

<table>
<thead>
<tr>
<th></th>
<th>Sparse Formats</th>
<th>Reshape</th>
<th>Concatenate</th>
<th>Slice</th>
<th>Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>scipy.sparse</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Looplets</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>TACO</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Burrito</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Program Synthesis is a Powerful Technique!

Program Specification → Program Synthesis →

0 | input x : BV[16]
1 | input y : BV[16]
2 | input z : BV[16]
3 | r0 = ir.mul(y,z)
4 | r1 = ir.add(x,r0)
5 | return r1
Satisfiability Modulo Theory (SMT) Solvers Enable Efficient Program Synthesis
Using Program Synthesis in AHA?
AHA Goal: Automatically Generate CGRA Compiler

- Specification of PE
- Specification of IR

Generate

Compiler

Application

- Lower to IR
- Instruction Selection
- Scheduling + Allocation
- Assembly

CGRA Executable
Use Program Synthesis to Generate Instruction Selection Rewrite Rules!

Specification of PE

Specification of IR

Program Synthesis

Lower to IR

Instruction Selection

Scheduling + Allocation

Assembly

CGRA Executable
Brand New Program Synthesis Technique!

Completeness Guaranteed!
Optimizations Provide Massive Speedups!

10x Speedup!

92x Speedup!
Efficiently Synthesizing Lowest Cost Rewrite Rules for Instruction Selection

Ross Daly
Cascade: An Application Pipelining Toolkit for Coarse-Grained Reconfigurable Arrays

Existing CGRA compilers do not produce high performance and energy efficient applications
• They lack pipelining resulting in low performance or exhaustively pipelining resulting in high power

Cascade is an end-to-end compiler which has:
• An automatic CGRA timing model generator
• A static timing analysis tool for CGRA applications
• A large set of existing and novel pipelining techniques integrated into an end-to-end flow

Stanford University
Improving the Performance of Convolutional Neural Networks on CGRAs

Yuchen Mei
Dedicated PnR & Pipelining for CNN on CGRAs

Try to solve limitations of CNN on CGRAs:

by:

- Systolic Array Placement
- PnR-optimized Pipelining
- MEM Rescheduling

Stanford University
Finch: A Placement and Routing Visualization and Editing Tool for Coarse-Grained Reconfigurable Arrays

Zhouhua Xie, Kalhan Koul, Jackson Melchert, Priyanka Raina

AHA Retreat 2023 Lightning Talk
Current Challenge

Difficulty in Place & Route (P&R) for CGRA physical design:

- **Inhomogeneous** resource distribution
- **Limited** resources on placement and routing
- **Complex** design space

Imperfect P&R Algorithm for CGRA

Problem Example: Scattered Placement
Primary Use Cases

• Editing for Last Mile Improvement
• Design Analysis
• Application Design Prototyping
• P&R Algorithm Analysis
Incorporating Formal Translation Validation into CGRA Compilers

- AHA tools have enabled agile hardware/software codesign
- Verification is still painful and a better solution exists:
  
  Formal translation validation of our CGRA compiler

- Will enable faster, more extensive verification that decreases debugging time and increases productivity
We present *Self-Driven Strategy Learning*, an online learning method for automated reasoning tasks that involve solving a set of related problem.

Executions of Bounded Model Checking with and without SDSL

SDSL invests time learning a good solving strategy in the beginning, which results in better performance when solving later problems.

**Lightweight Online Learning for Sets of Related Problems in Automated Reasoning**

- Haoze (Andrew) Wu
  Stanford Univ.
- Christopher Hahn
  Stanford Univ.
- Florian Lonsing
  Unaffiliated
- Makai Mann
  MIT Lincoln Lab
- Raghuram Ramanujan
  Davidson College
- Clark Barrett
  Stanford Univ.
An Abstract Calculus for Optimization Modulo Theories

Nestan Tsiskaridze¹
Clark Barrett¹
Cesare Tinelli²
¹ Stanford University
² The University of Iowa

Applications
Scheduling/Planning with Resources
Requirements Engineering/Specification Synthesis
System Design/Configuration
Formal Verification/Model Checking
Program Analysis  Security Analysis
Machine Learning
Quantum Annealing
...

Satisfiability Modulo Theories (SMT)
Powerful Search Engines
The Abstract DPLL(T) Calculus for SMT:
Superior
Efficiency +
Expressiveness +
Flexibility

Optimization Modulo Theories (OMT)
An Abstract OMT Calculus:
A foundation for theoretical understanding and research
A blueprint for practical implementations

Find me to Learn HOW!

Rapid Integration of Flow IP with Agile Physical Design Tools

Alex Carsello, Christopher Torng, Mark Horowitz
Agile PD Tools Struggle to Facilitate Reuse at Scale

- Agile PD Tools aim to reduce PD effort by providing reusable modular flows with customization for design/tech
  - Mflowgen, Silicon Compiler, HAMMER

Small-Scale reuse:
- Small teams with established communication channels
- Limited scope/complexity
- Existing tools work well

Reuse at scale:
- Node produced by one team and used by another
- Little-to-no communication
- Different projects/complexity
- Effort required to use IP may exceed effort required to develop your own
Testing is the Answer

- Augment mflowgen Node primitive with tests
  - Tests can attach to any PD flow graph
- Test specify where in the flow they need to run
- Tests tell mflowgen where it can reduce effort or skip steps entirely to get test results faster
- Tests can span multiple hierarchies/subgraphs
- Talk to me at my poster to learn more!
Avanergy: An Architecture-VLSI Abstraction for Automated Energy-Aware Design-Space Exploration Tools
Avanergy: An Architecture-VLSI Abstraction for Automated Energy-Aware Design-Space Exploration Tools

a very different world of hardware design
Avanergy: An Architecture-VLSI Abstraction for Automated Energy-Aware Design-Space Exploration Tools

A very different world of hardware design

1. Hardware experts with ~5-20+ years of experience
   - CPU
   - TPU
   - Accelerators

   ... who are also
   domain enthusiasts with ~1 year or less of experience
Avanergy: An Architecture-VLSI Abstraction for Automated Energy-Aware Design-Space Exploration Tools

a very different world of hardware design

A specific way that things have always been done

1. Hardware experts with ~5-20+ years of experience

   CPU
   TPU
   Accelerators

   ... who are also

   domain enthusiasts with ~1 year or less of experience
Avanergy: An Architecture-VLSI Abstraction for Automated Energy-Aware Design-Space Exploration Tools

a very different world of hardware design

A specific way that things have always been done

- Specification
- RTL Design
- Verification
- Physical Design and Impl
- Tapeout

Hardware experts with ~5-20+ years of experience

CPU

TPU

Accelerators

... who are also domain enthusiasts with ~1 year or less of experience

A specific design methodology

- Software Stack
- Architecture
- VLSI
- Technology
Avanergy: An Architecture-VLSI Abstraction for Automated Energy-Aware Design-Space Exploration Tools

a very different world of hardware design
Avanergy: An Architecture-VLSI Abstraction for Automated Energy-Aware Design-Space Exploration Tools

A very different world of hardware design

2

Application domain experts
with ~1 year or less of experience with hardware design

Domain Experts

... whose primary jobs are
Machine Learning
Image Processing
Video Coding
Cryptography
Wireless

2
Avanergy: An Architecture-VLSI Abstraction for Automated Energy-Aware Design-Space Exploration Tools

A very different world of hardware design

2

Key Question: Do our existing methods work?

Specification

RTL Design

Verification

Physical Design and Impl

Development Time

Tapeout

Domain Experts

... whose primary jobs are

Machine Learning
Image Processing
Video Coding
Cryptography
Wireless

Application domain experts

with ~1 year or less of experience with hardware design
Avanergy: An Architecture-VLSI Abstraction for Automated Energy-Aware Design-Space Exploration Tools

a very different world of hardware design

Key Question: Do our existing methods work?

Specification

RTL Design

Verification

Physical Design and Impl

Development Time

Tapeout

Domain Experts

... whose primary jobs are

Machine Learning
Image Processing
Video Coding
Cryptography
Wireless

Application domain experts

with ~1 year or less of experience with hardware design

Software Stack

Architecture

VLSI

Technology

Does the stack need to change to "hide" more?
Avanergy: An Architecture-VLSI Abstraction for Automated Energy-Aware Design-Space Exploration Tools

a very different world of hardware design

Application domain experts
with ~1 year or less of experience with hardware design

... whose primary jobs are
Machine Learning
Image Processing
Video Coding
Cryptography
Wireless

Avanergy is a new abstraction that numerically systematizes energy-aware design-space exploration in a way that future automated tools can build upon it
Automatic Discovery of Late Stage Timing Bugs

Raj Setaluri, Christopher Torng

Complex chips have timing bugs

- **CPU**
- **Global Buffer**
- **CGRA**

Amber

Timing bugs still exist!

Why do these bugs happen?

```
set_multicycle_path 3
  -from Q0  -to Q1  -setup
```

- **Static timing analysis** ✓
- **Gate-level simulation** ✘

Timing bugs are needles in a haystack

Automatically find timing bugs using formal tools

- **Complex SoC Designs**
- **Timing-Sensitive Interfaces**
- **Small, Automated, and Targeted Gate-Level Simulations**

Human error in constraints (static timing analysis is clean)
ASPEN: Acceleration of Visual-Inertial Odometry for Extended Reality on an FPGA

Kathleen Feng

Stanford University

30 August 2023
Extended Reality Pipeline

Perception Pipeline
- Hand/Eye Tracking
- Visual-Inertial Odometry
- Scene Reconstruction

Visual Pipeline
- Application Rendering
- Post-Processing
- Display

Audio Pipeline
- Recording
- Encoding
- Playback

Camera, IMUs, Depth Camera
Mic
CPU Performance Breakdown

Source: ILLIXR [Huzaifa 2021]
VIO: Visual Inertial Odometry

Calculates 3D user position from sensors
- IMUs, cameras \( (x, y, z, \theta, \phi, \psi) \)
- Most dominating subtask, represents \( \sim 40\% \) of XR workload

Using OpenVINS as gold model [https://docs.openvins.com/index.html]
A Fast Large-Integer Extended GCD Algorithm and Hardware Design

Kavya Sreedhar, Mark Horowitz, Christopher Torng
A Fast Large-Integer Extended GCD Algorithm and Hardware Design

Kavya Sreedhar, Mark Horowitz, Christopher Torng

Target Platform

XGCD Design Space

Algorithm Choice

Application Requirements
A Fast Large-Integer Extended GCD Algorithm and Hardware Design

Kavya Sreedhar, Mark Horowitz, Christopher Torng
Interrupt-driven MLSD-based Links

Zach Myers, Stanford VLSI Group

\[ s_i \rightarrow \text{Channel} \rightarrow \text{Linear Equalization} \rightarrow \text{Nonlinear Equalization} \rightarrow s_i \]

\[ s_i \rightarrow \text{Channel} \rightarrow \text{Linear EQ} \rightarrow \text{Error Checker} \rightarrow \text{Fork-on-Flag} \rightarrow \text{FIFO} \rightarrow \text{Error Corrector} \rightarrow \text{Combine} \rightarrow s_i \]
Co-Designing AI Models and Embedded DRAMs for Efficient On-Device ML Training

Thierry Tambe
Data transience is pervasive

Weights are **transient**, Activations are **transient**, Gradients are **transient**

Computational graph of DL training
eDRAM as main on-chip storage medium

+ Denser than SRAM
+ Lower access energy than SRAM
+ Can be made multi-level

- Activations must be buffered for duration of fwd and bwd passes
- Retention time in order of a few microseconds
Algorithm-System Co-Design for eDRAM-based Computing

- Algorithms
- HW Architecture

On-Chip ML Training with Refresh-Free eDRAMs

- eDRAM Architecture
Reversible neural networks to avoid buffering activations

Original DNN

Reversible DNN

Equations during the training process

Forward pass:
\[ y_2 = F_1(x_1) + x_2 \]
\[ y_1 = F_2(y_2) + x_1 \]

Backward Pass:
\[ x_1 = y_1 - F_2(y_2) \]
\[ x_2 = y_2 - F_1(x_1) \]

Activations \( x_1 \) and \( x_2 \) are recomputed instead of being loaded from mem
Interleaved memory access patterns to promote implicit refresh
ASIC co-design to minimize data lifetime
Thank you!