

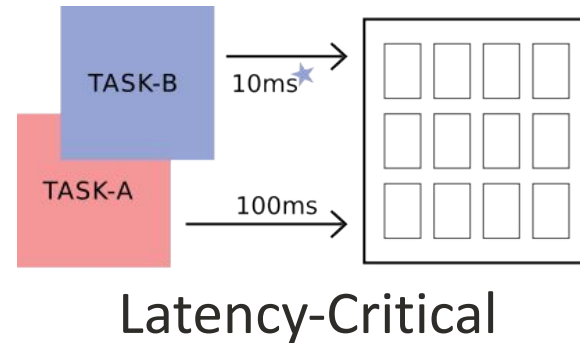
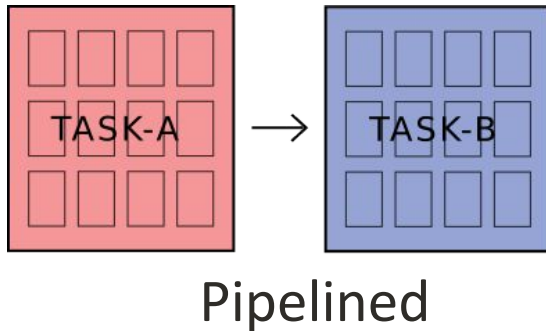
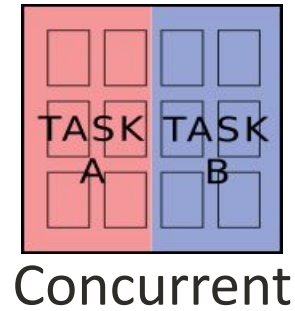
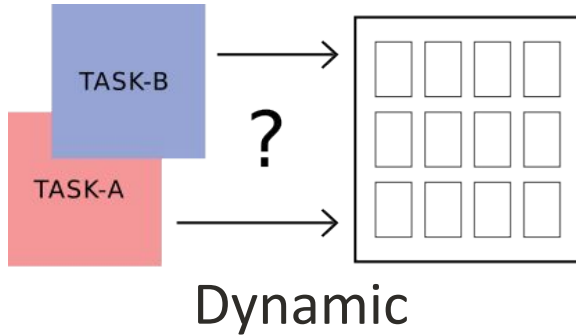
Using a CGRA with Dynamic Partial Reconfiguration

Taeyoung Kong

Kalhan Koul, Priyanka Raina, Mark Horowitz, and Christopher Torng

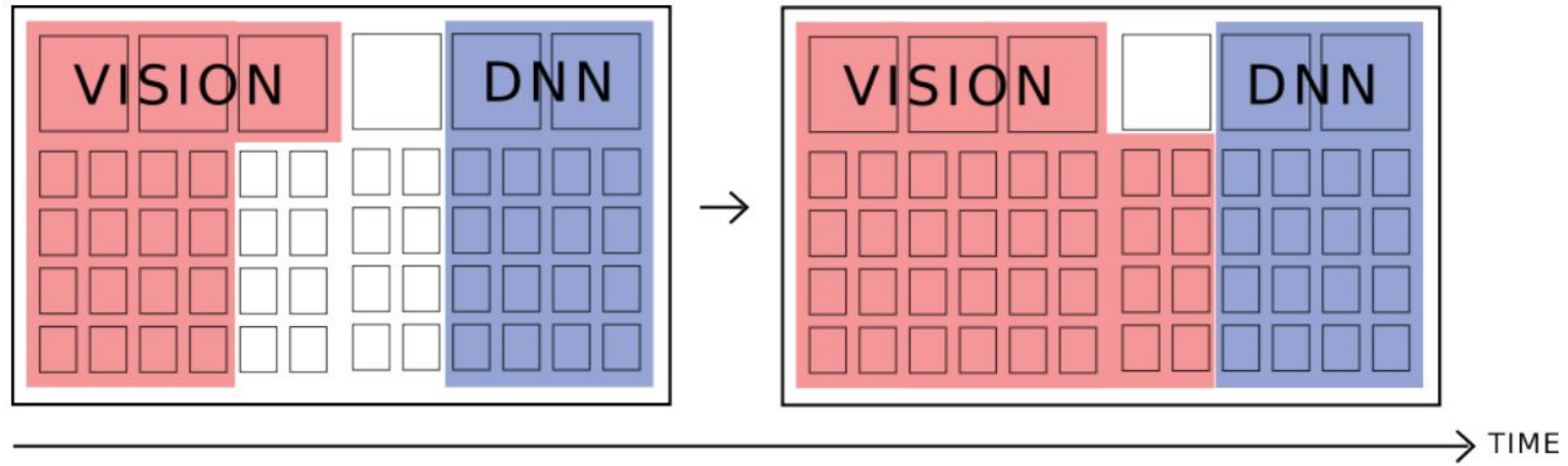
Stanford University

Motivation - Multi-Task Workloads



=> Exploit *Dynamic Partial Reconfiguration* of CGRA to adapt to such workloads

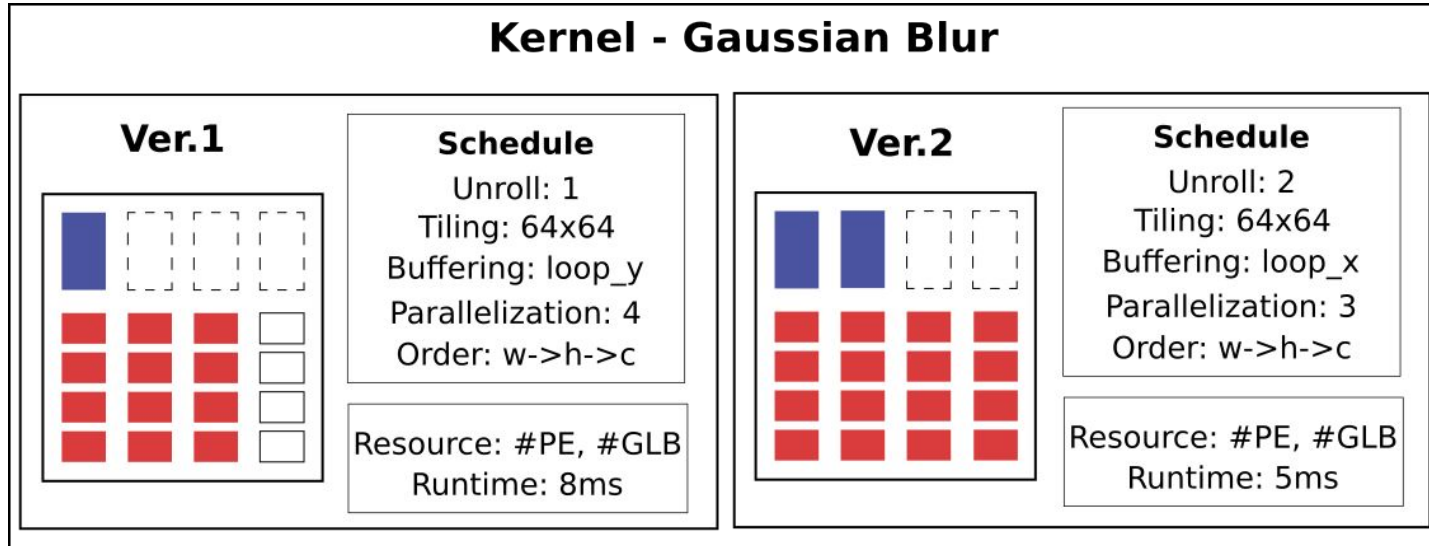
Architecture - Fast DPR & Flexible Resource Allocation



Fast DPR allows a rapid change of underlying kernel

Flexible resource allocation of MEM units and PE units increases resource utilization

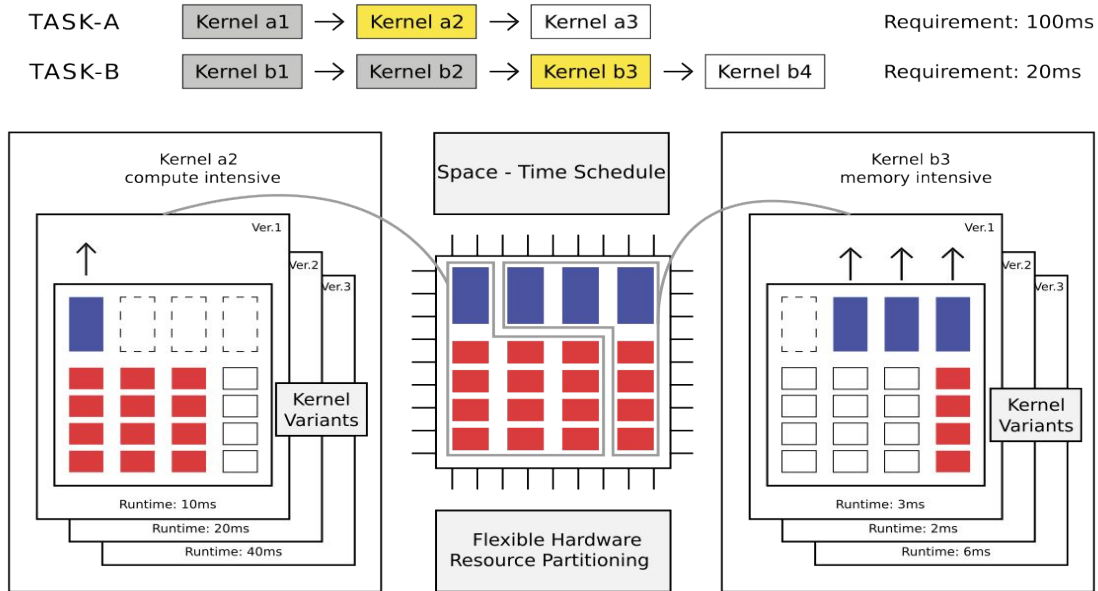
Compiler - Various Kernel Variants



Compiler prepares several versions of the single kernel (Variants)

- different hardware resources
- different runtime

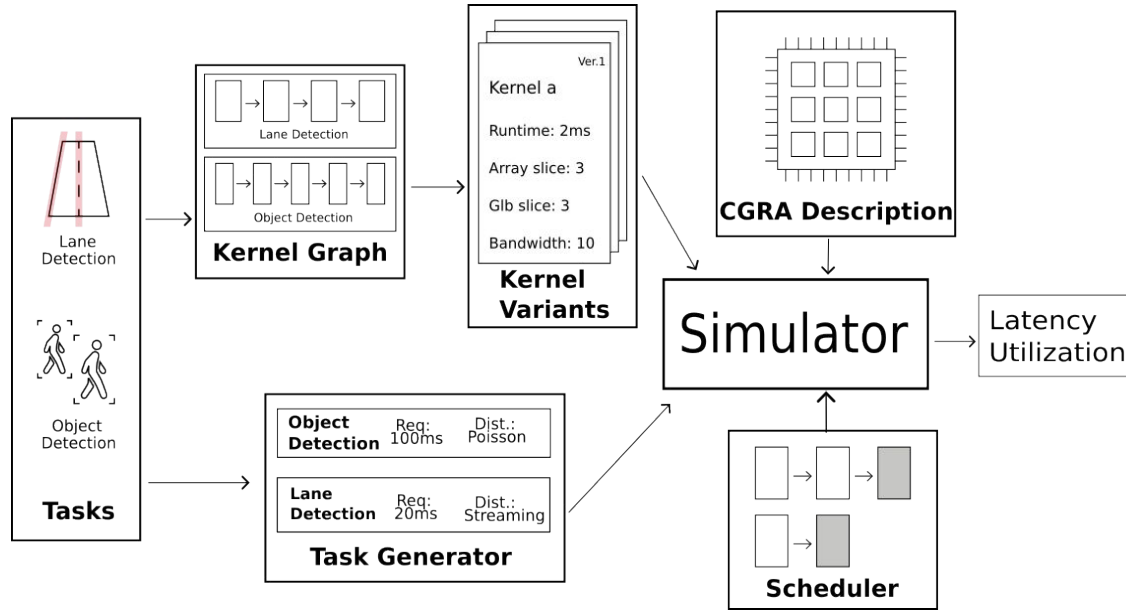
Runtime - Resource-Aware Scheduler



Scheduler selects one of the kernel variants at runtime based on

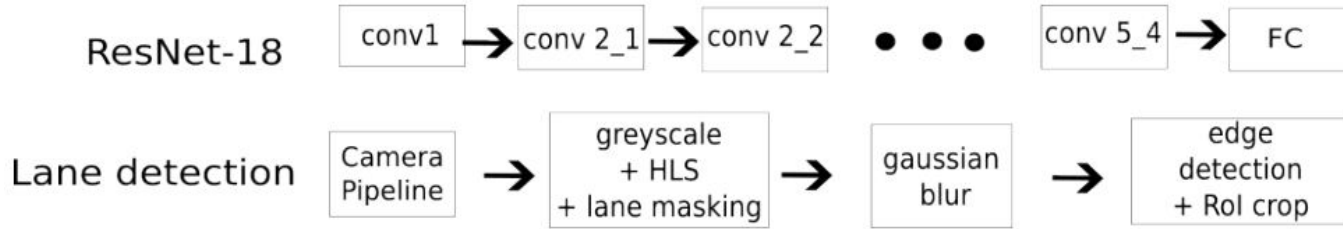
- remaining runtime requirements
- resource availability

Simulator - Python-Based Performance Simulator



Performance simulator allows easy exploration of CGRA and scheduler

Result - Image Processing + ML Workload



Lane detection + Object detection

1.6x speedup with

- DPR & flexible resource allocation
- resource-aware scheduler



Thank You

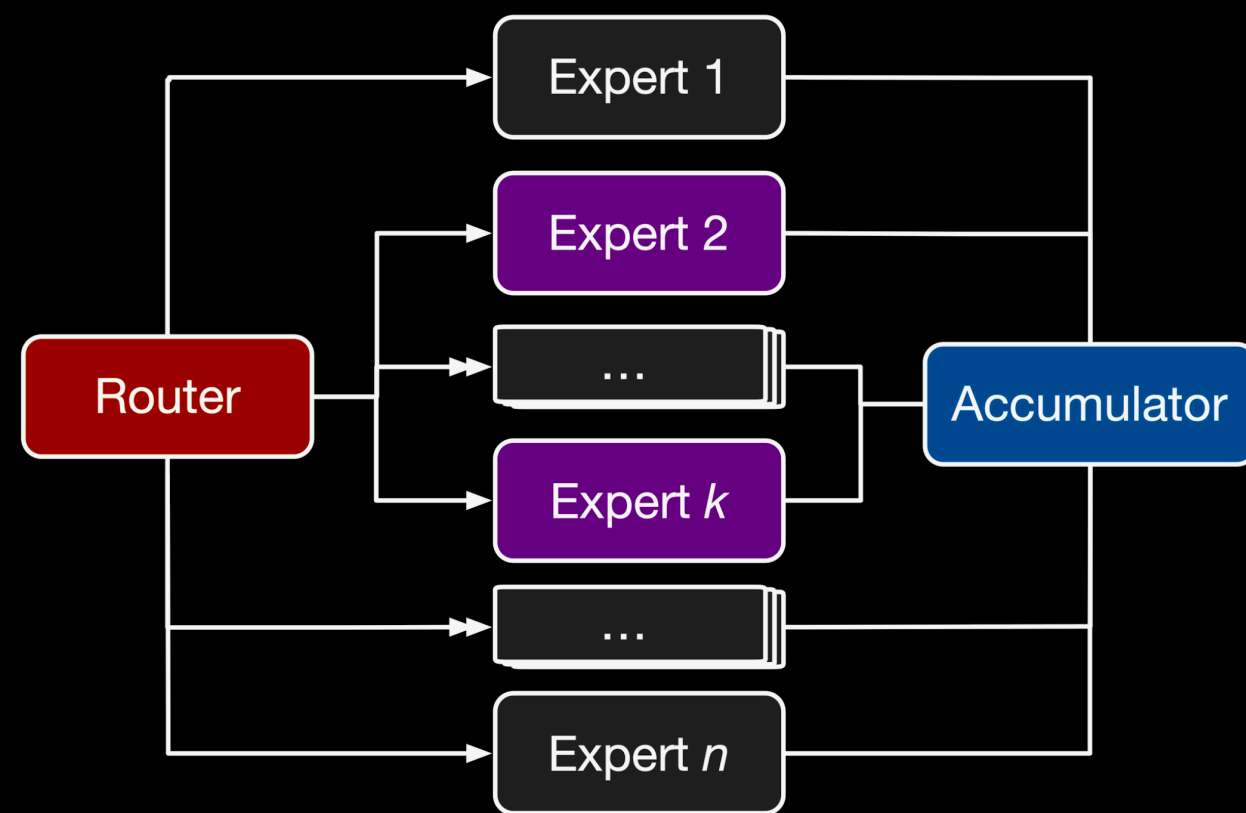
Scorch

A library for sparse machine learning

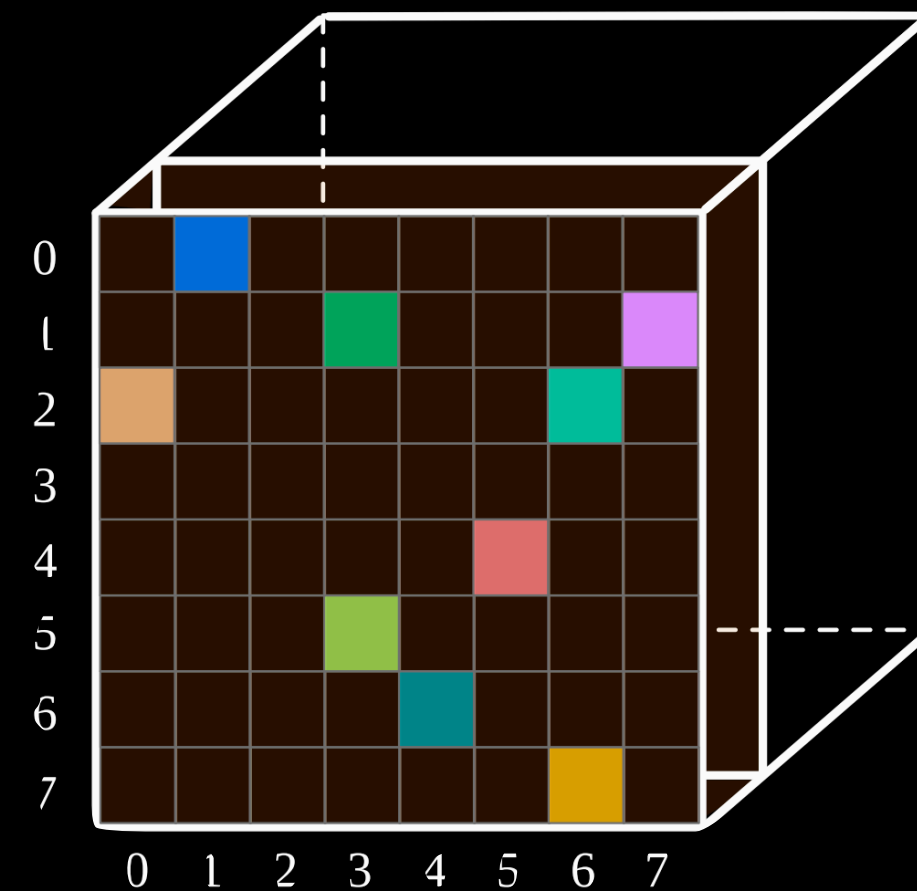
Bobby Yan*, Alexander J. Root*, Trevor Gale*[†], Fredrik Kjolstad*

*Stanford University, [†]Google Research

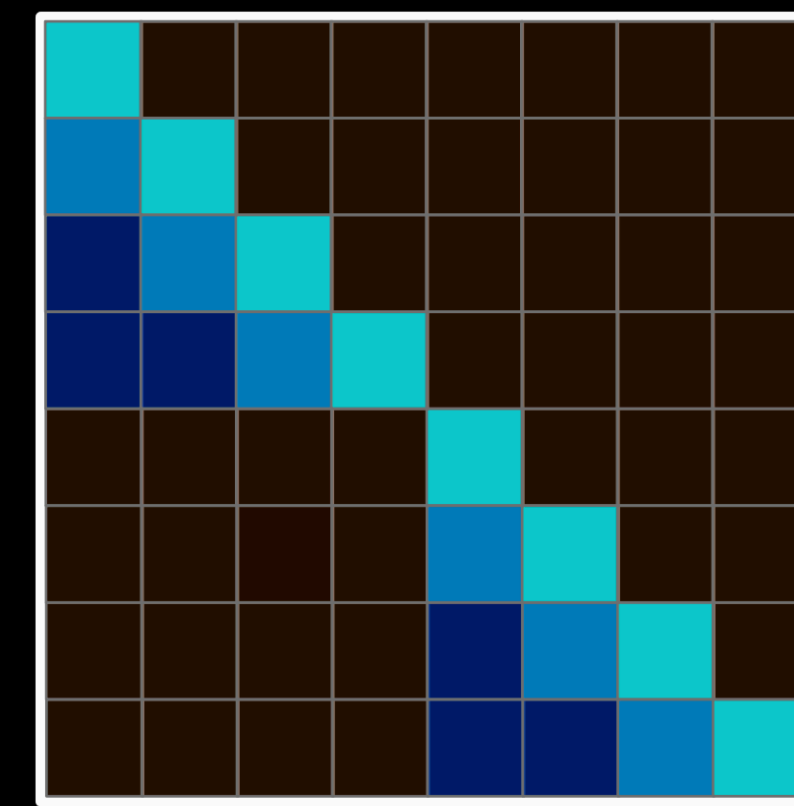
Sparsity comes from data and model design



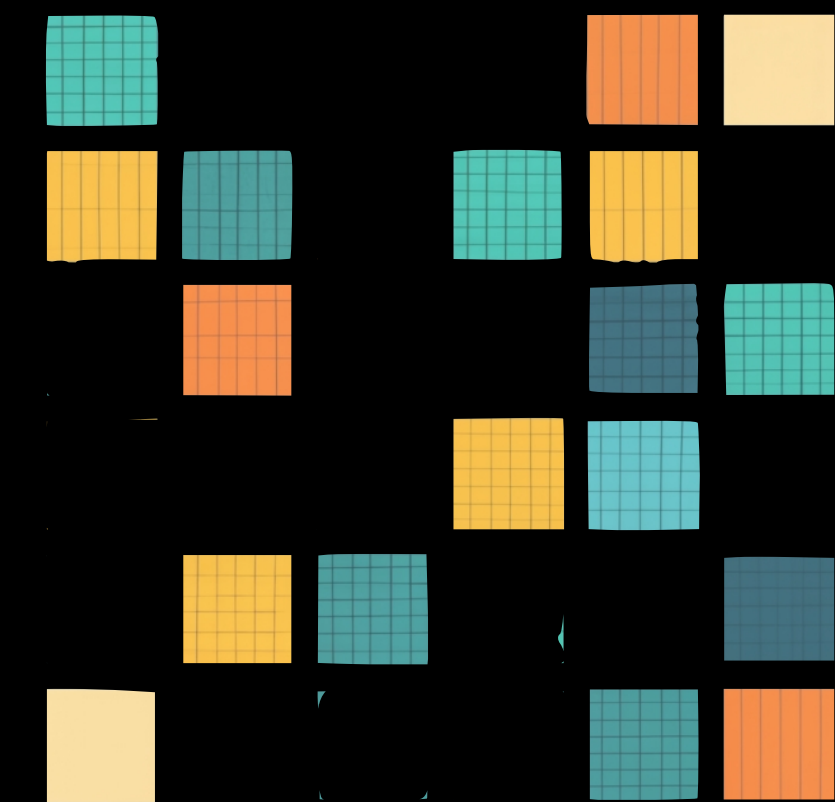
Mixture of experts



Graph neural networks



Sparse transformers



Recommender systems

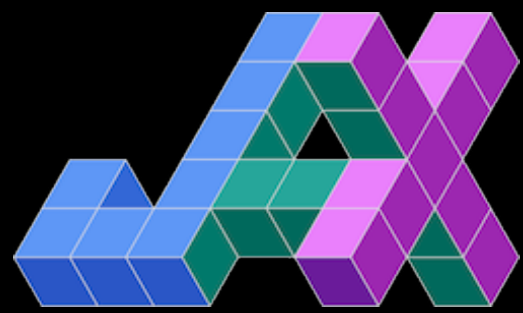
Programming model landscape is fragmented



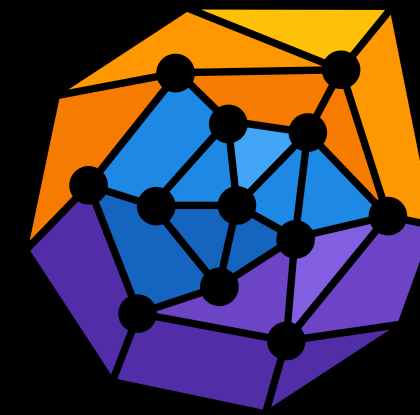
torch.sparse



tf.sparse



jax.sparse



PyG



DGL



TVM



MLIR Sparse

cuSPARSE

cuSPARSE

intel®

MKL-DNN

Programming model landscape can be unified



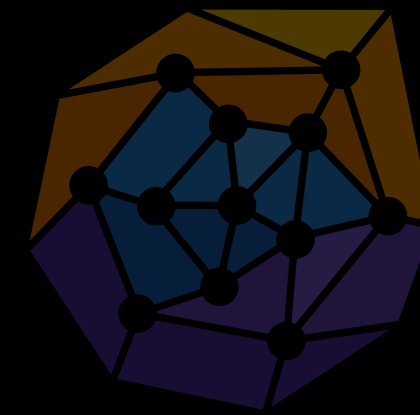
torch.sparse



tf.sparse



jax.sparse



PyG



DGL



Scorch



TVM



MLIR Sparse

cuSPARSE

cuSPARSE

intel®

MKL-DNN

Compiling Declarative Recurrences To Imperative Code

Shiv Sundram, Muhammad Usman Tariq,
Fred Kjolstad

August 2023

Three classes of algorithms

1 Linear Solvers
(LU, Cholesky)

2

3

Three classes of algorithms

- 1 Linear Solvers
(LU, Cholesky)
- 2 Dynamic Programs
(Sequence Alignment, Bellman)
- 3

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

$$L_{ij} = (A_{ij} - \sum_{k=0}^j L_{ik}L_{jk}) / L_{jj}$$
$$: j < i$$

$$L_{ij} = \sqrt{A_{ij} - \sum_{k=0}^j L_{jk}L_{jk}}$$
$$: j = i$$

All expressible in a common language

Recurrence Equations

$$L_{ij} = (A_{ij} - \sum_{k=0}^j L_{ik}L_{jk}) / L_{jj}$$

$: j < i$

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

$: j = i$

All expressible in a common language

Recurrence Equations

$$L_{ij} = (A_{ij} - \sum_{k=0}^j L_{ik}L_{jk}) / L_{jj}$$

$: j < i$

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

$: j = i$

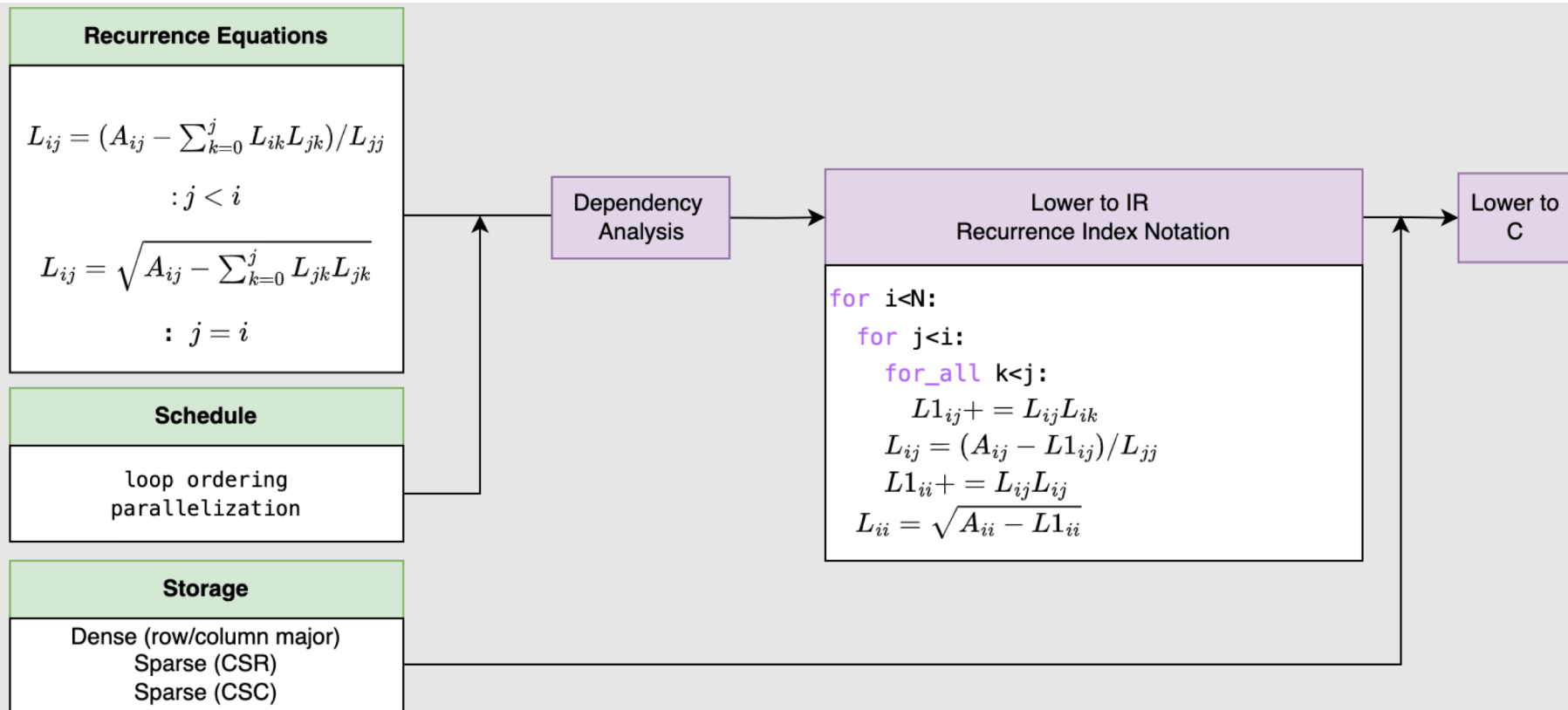
Schedule

loop ordering
parallelization

Storage

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

All expressible in a common language



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



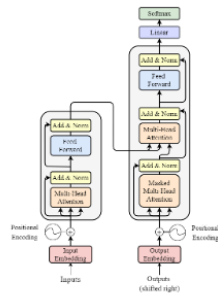
STANFORD

Expressing ML Applications in SAM

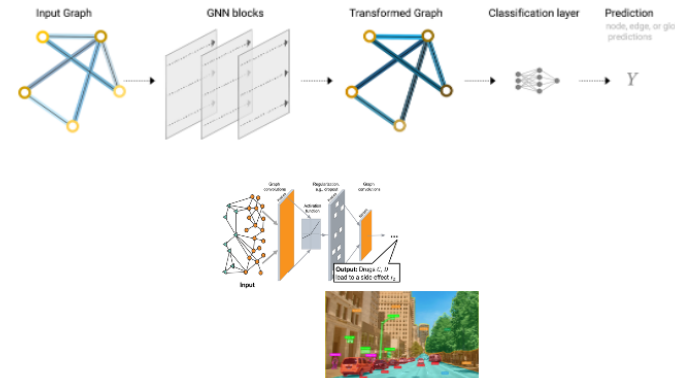
- Prior work, Sparse Abstract Machine (SAM), is not enough to express sparse ML

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

Transformers



GNNs

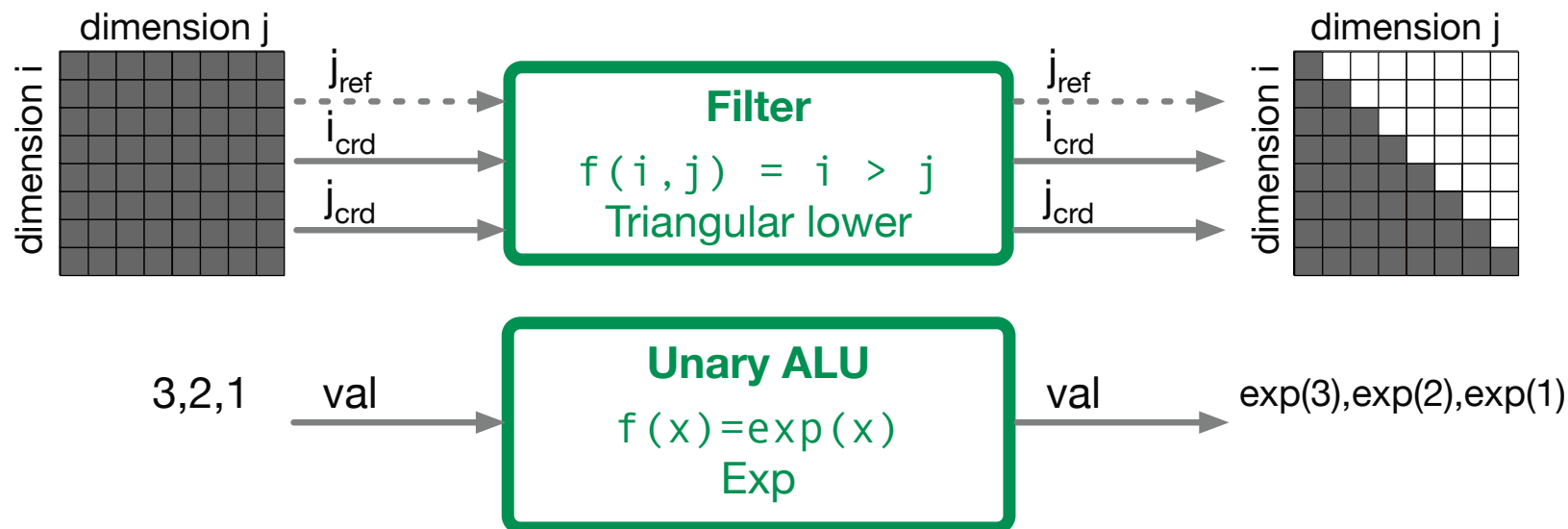


- 3-Tensor multiplication ✓
- 4-Tensor multiplication ✓
- Tensor addition ✓
- Slicing ✗
- Reshape/transpose/split/concat ✗
- Array programming ✗
- Data/mask generator ✗

- SpMM ✓
 - SDDMM ✓
 - SpMV ✓
 - Slicing ✗
 - Reshape/transpose ✗
 - Data/mask generator ✗
- ✓ 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

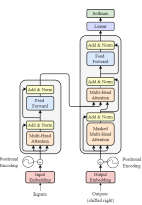


Expressing ML Applications in SAM

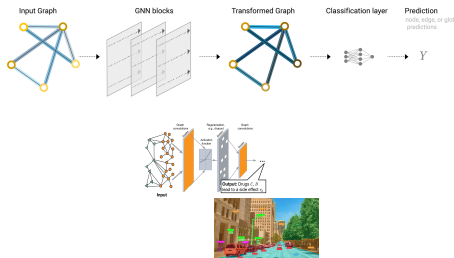
So now it can

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

Transformers



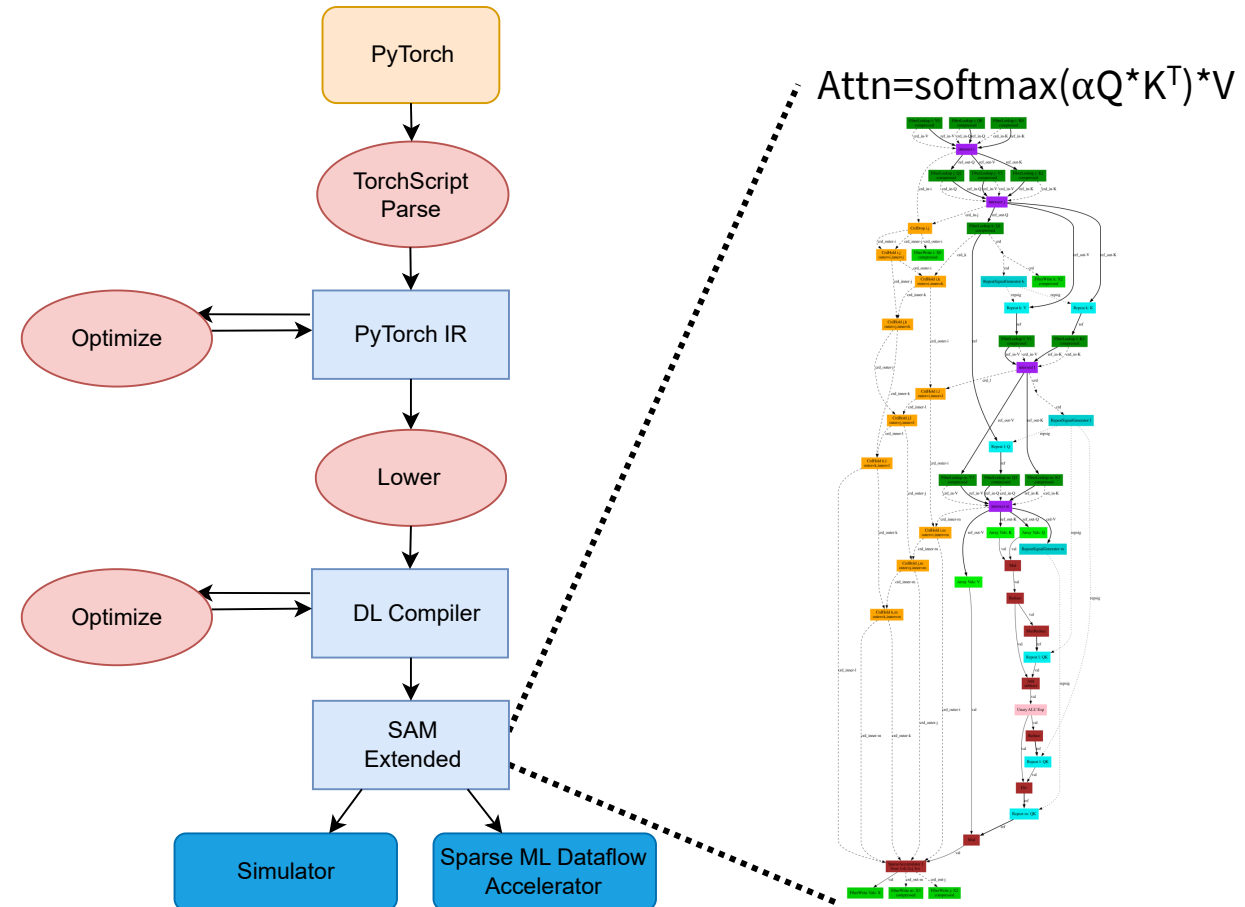
GNNs



- 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 ✓

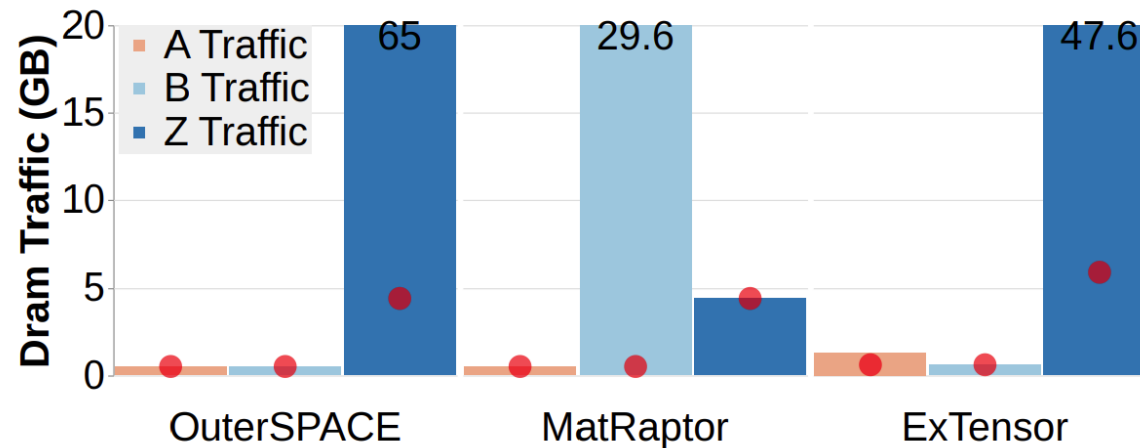
- ✓ Supported in SAM extended
- ✗ Not in SAM



Reuse and Traffic with Tiling for Sparse Accelerators

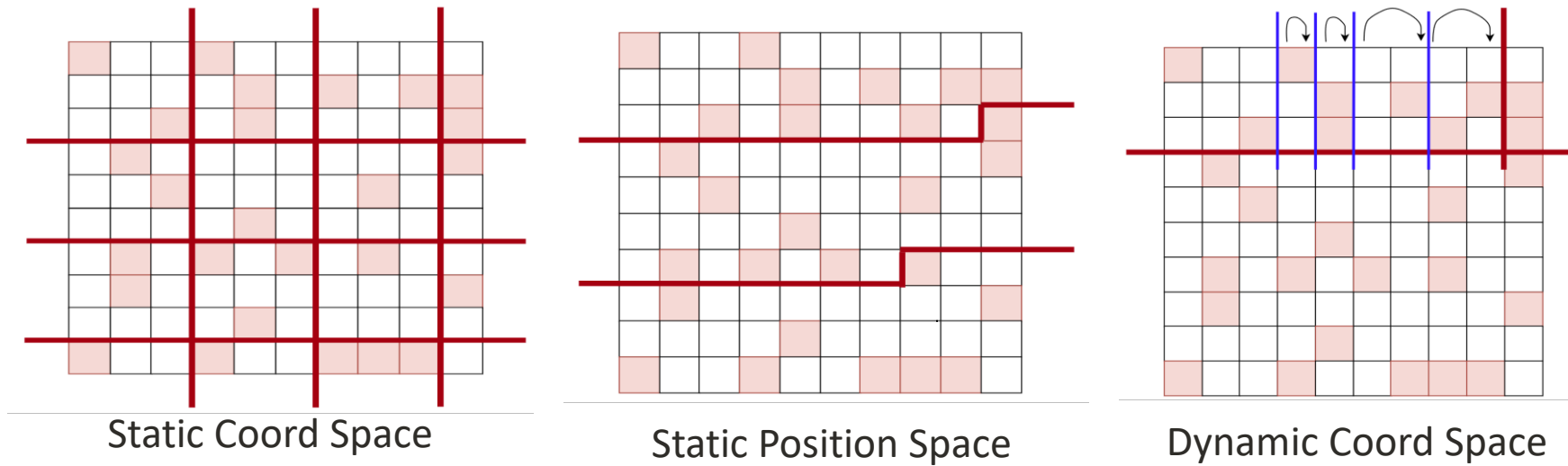
Ritvik Sharma, Olivia Hsu, Max Strange, Rubens Lacouture, Fred Kjolstad, Mark Horowitz

Reuse and Traffic with Tiling for Sparse Accelerators



A sparse accelerator's traffic vs its optimal memory traffic (red dots)

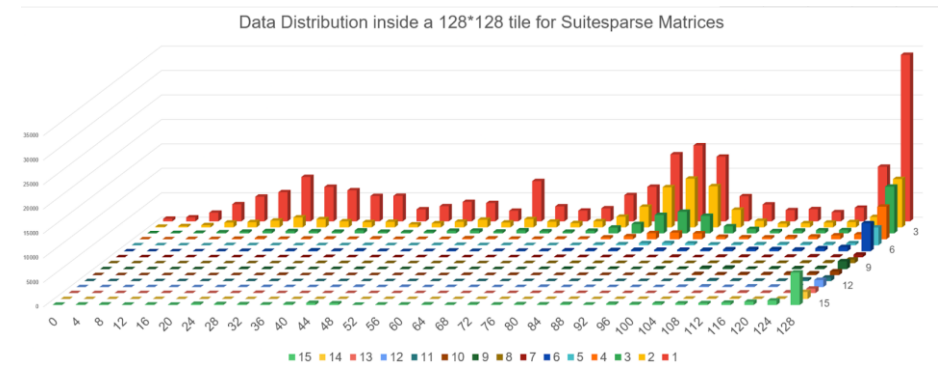
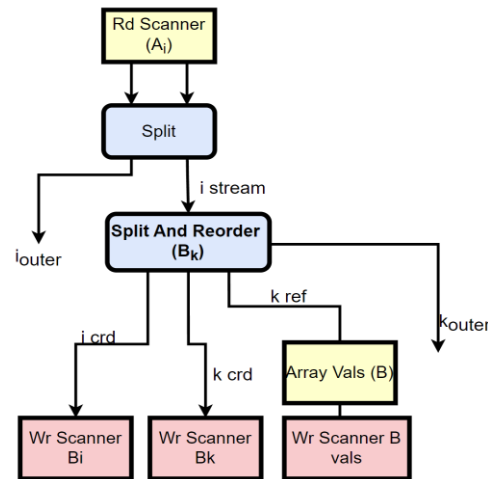
Reuse and Traffic with Tiling for Sparse Accelerators



Different Tiling Strategies

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



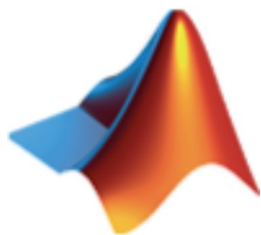
SAM primitives and graphs for Tiling

Suitesparse matrices data distribution

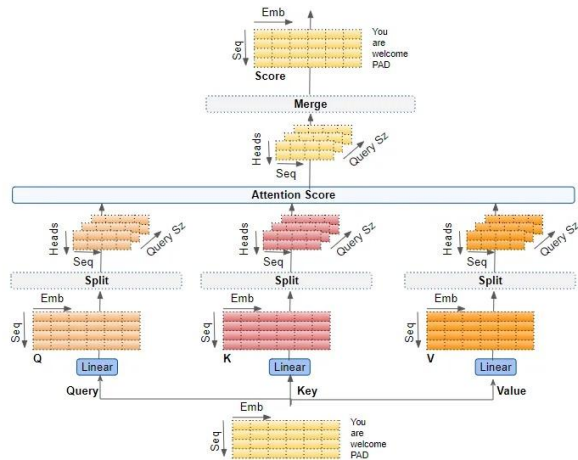
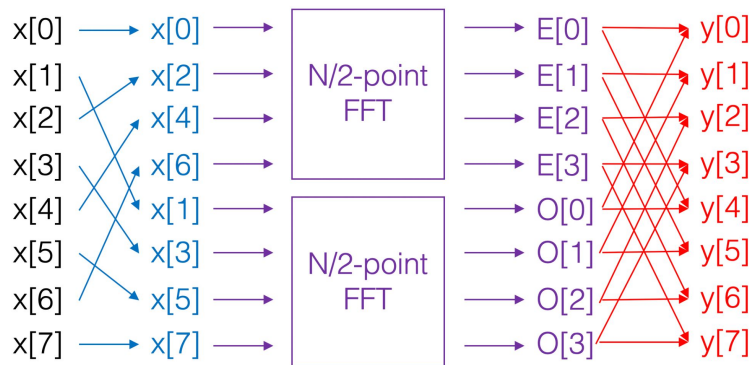
Sparse Shape Operators

Alexander J. Root, Bobby Yan, Peiming Liu, Aart Bik, Fredrik Kjolstad

Array programming is ubiquitous

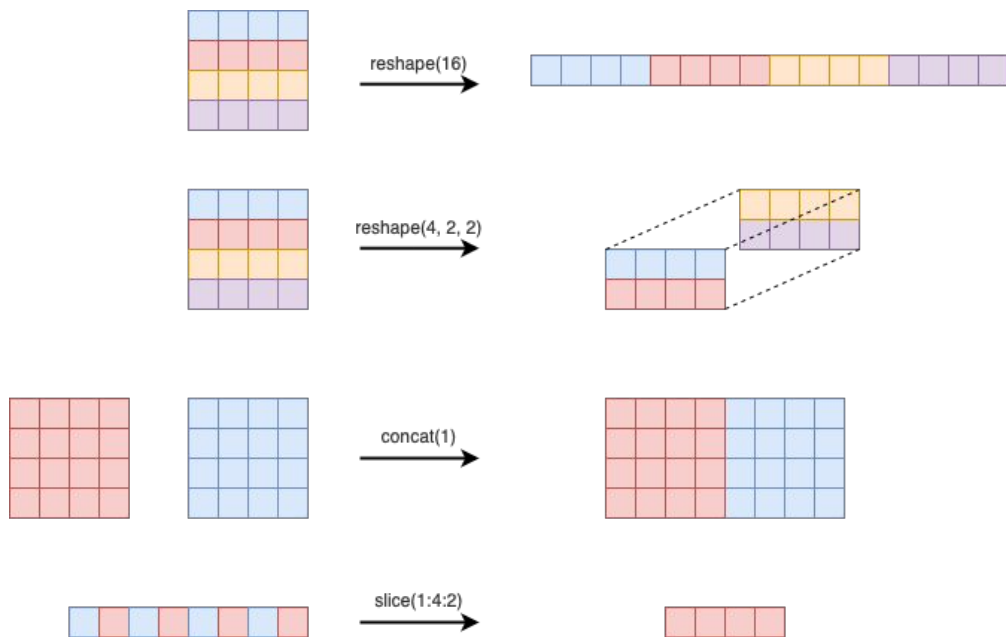


Array shapes are often manipulated



$$b = \begin{bmatrix} X & 0 & 0 \\ 0 & Y & 0 \\ 0 & 0 & Z \end{bmatrix} a$$

Shape operators



Sparse shape operator support is sparse























 PyTorch

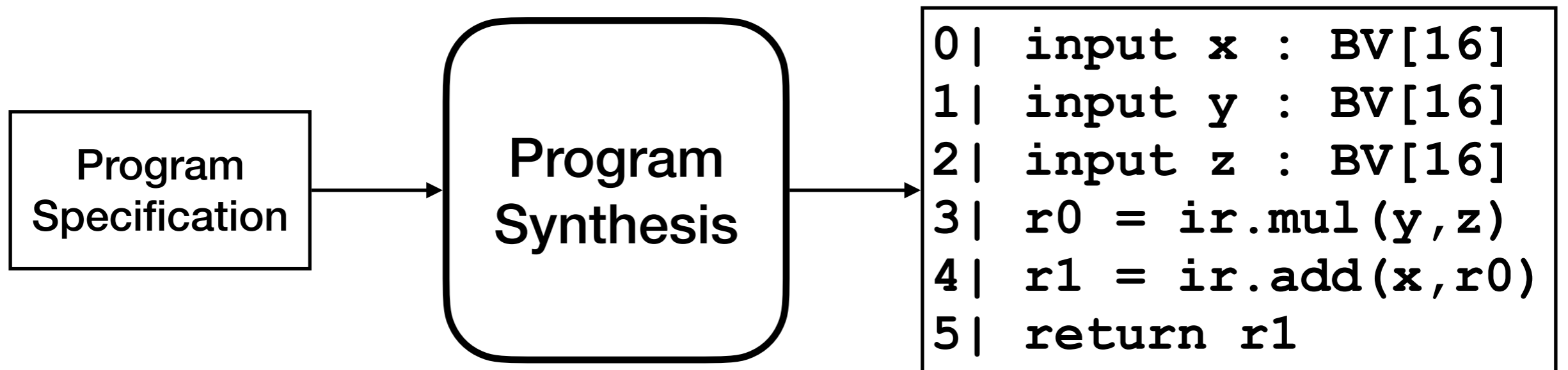


- Limited data structure support
- Lacking fusion across function calls

Sparse shape operator support is sparse

	Sparse Formats	Reshape	Concatenate	Slice	Fusion
scipy.sparse					
Looplets					
TACO					
Burrito					

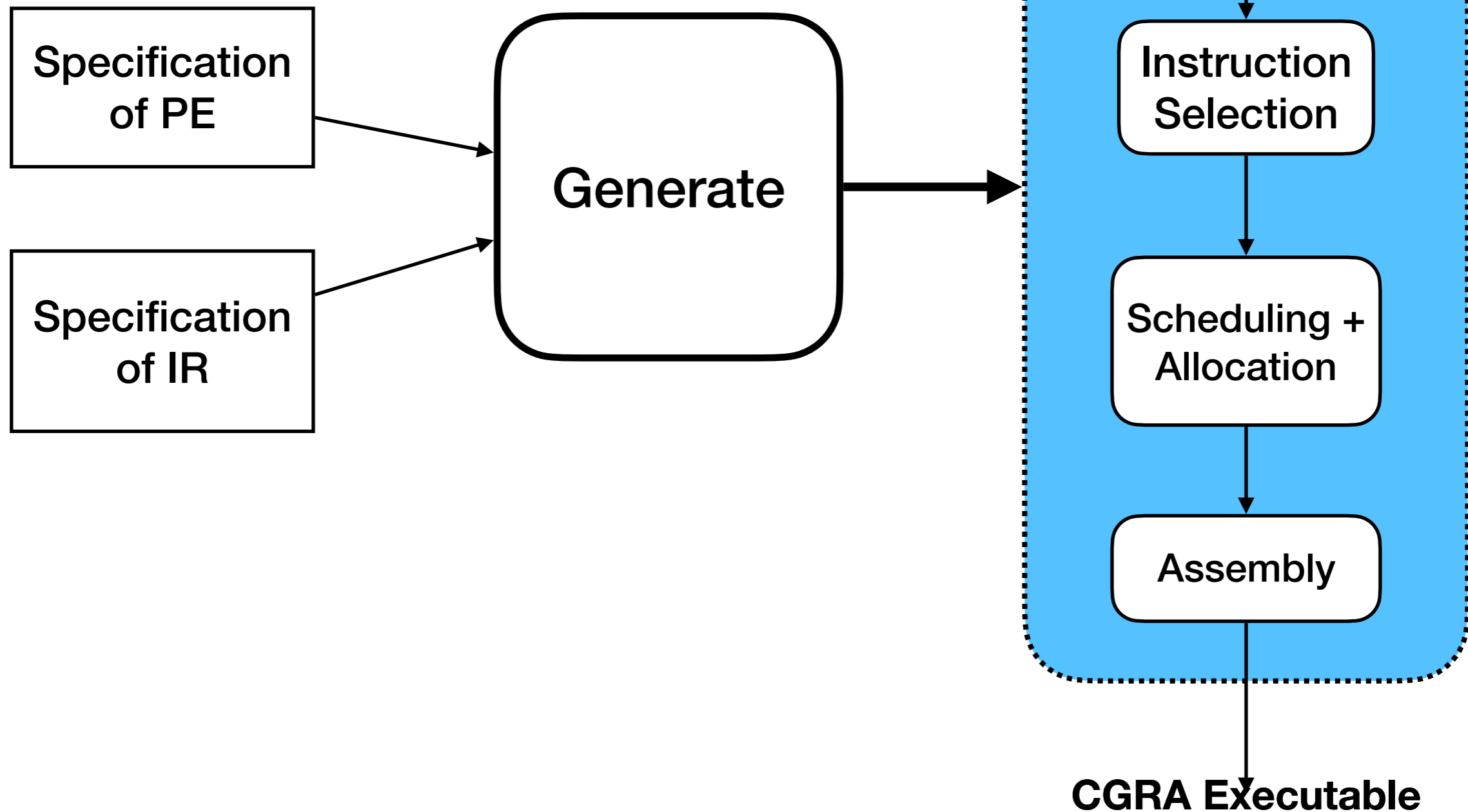
Program Synthesis is a Powerful Technique!



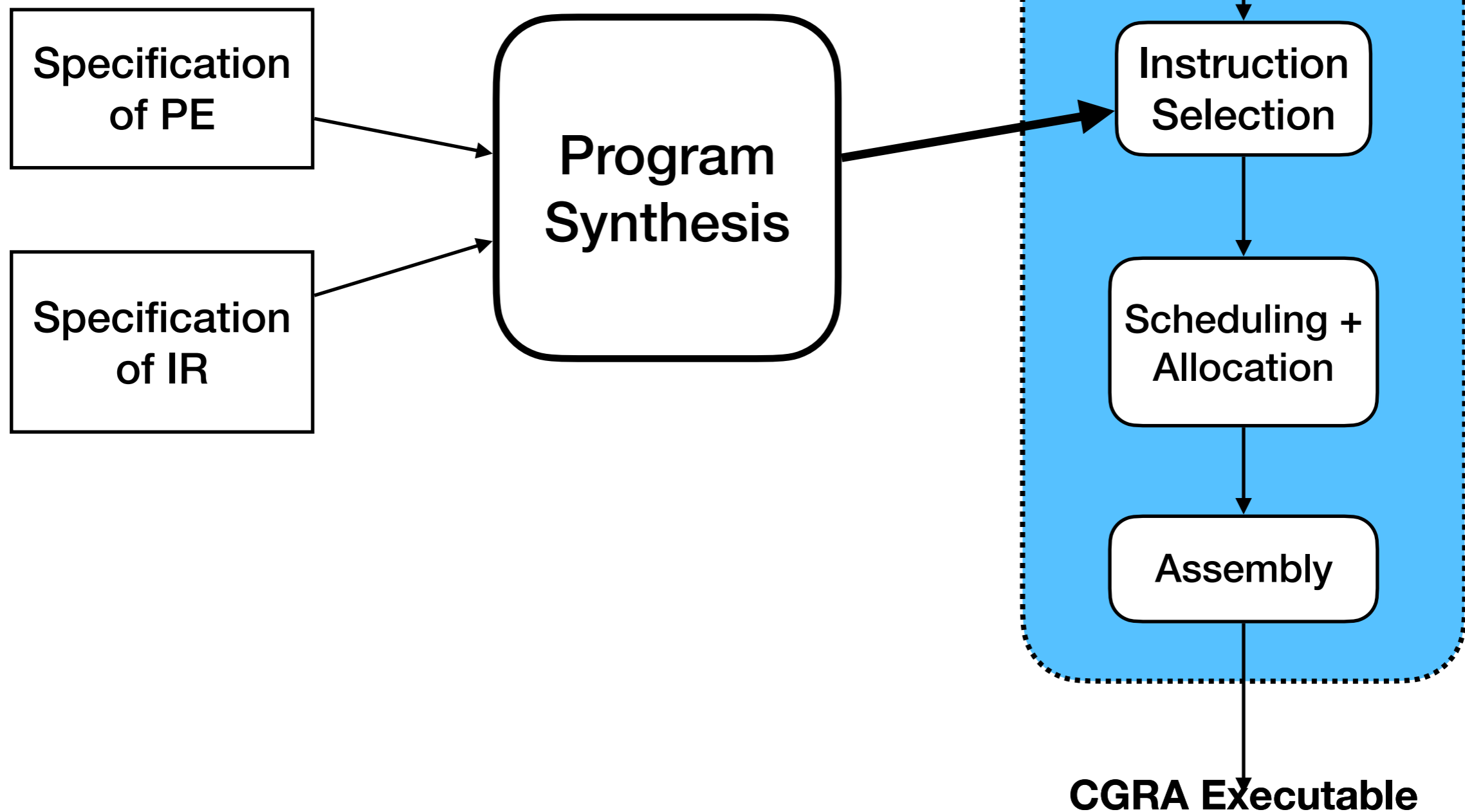
**Satisfiability Modulo Theory
(SMT) Solvers Enable
Efficient Program Synthesis**

Using Program Synthesis in AHA?

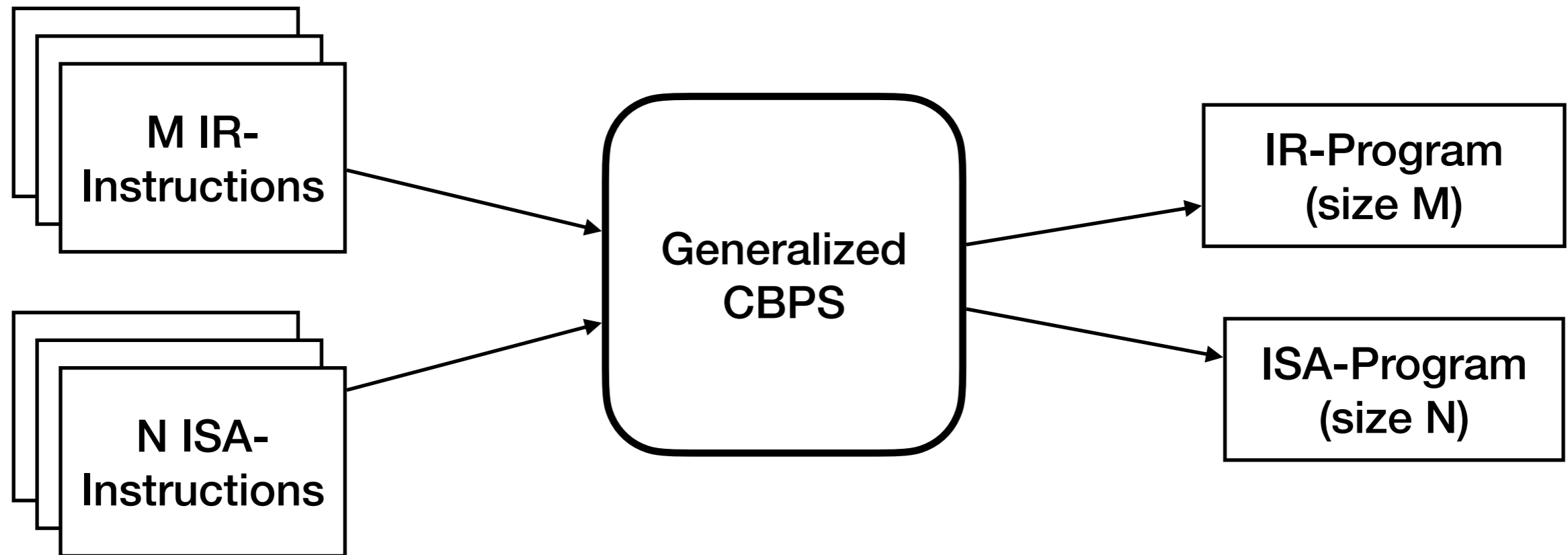
AHA Goal: Automatically Generate CGRA Compiler



Use Program Synthesis to Generate Instruction Selection Rewrite Rules!

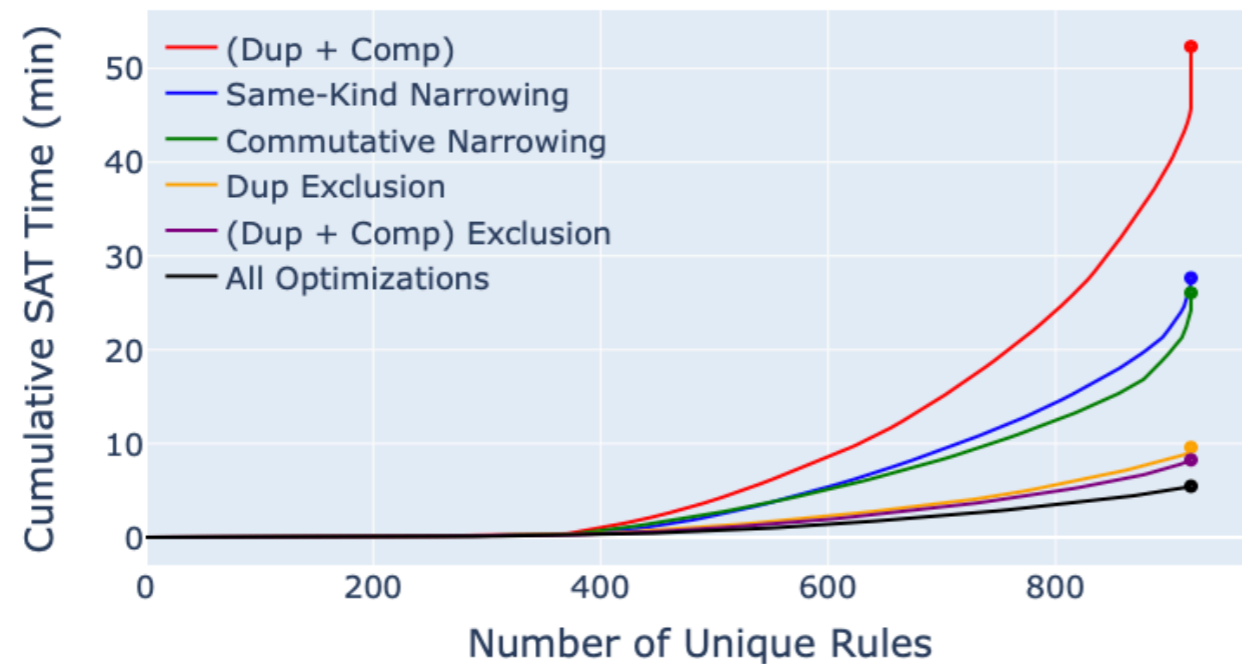


Brand New Program Synthesis Technique!



***Completeness* Guaranteed!**

Optimizations Provide Massive Speedups!



**10x
Speedup!**



**92x
Speedup!**

Efficiently Synthesizing Lowest Cost Rewrite Rules for Instruction Selection

Ross Daly

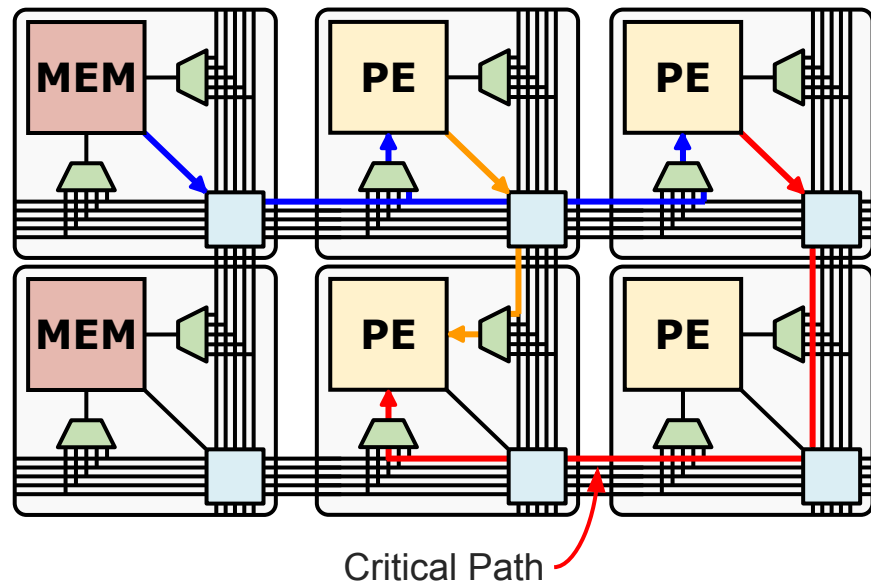
Cascade: An Application Pipelining Toolkit for Coarse-Grained Reconfigurable Arrays

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



Improving the Performance of Convolutional Neural Networks on CGRAs

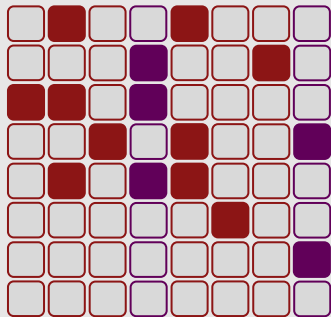
Yuchen Mei

Dedicated PnR & Pipelining for CNN on CGRAs

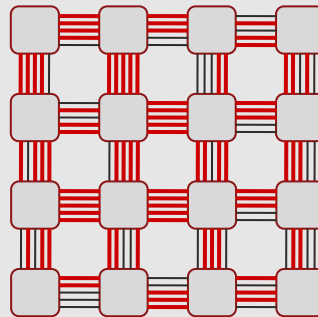
Try to solve limitations
of CNN on CGRAs:

by:

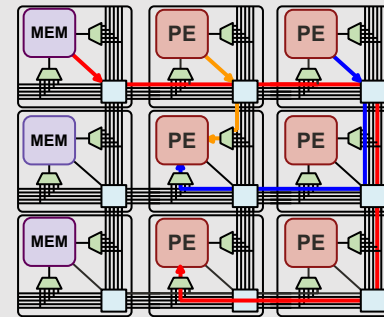
Low Utilization



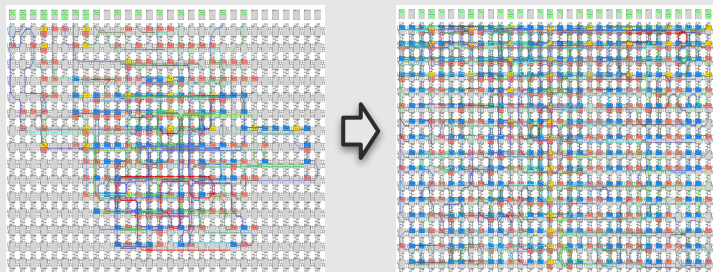
Poor Routability



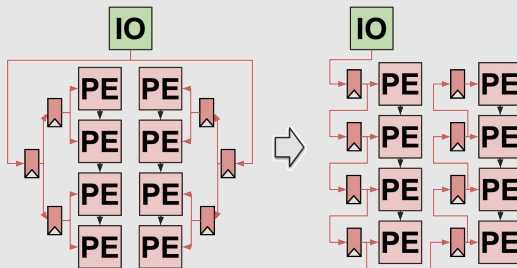
Long Critical Path



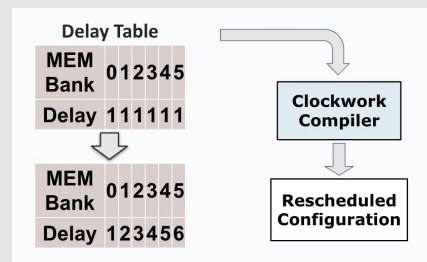
Systolic Array Placement



PnR-optimized Pipelining



MEM Rescheduling



Finch:

A Placement and Routing Visualization and Editing Tool for Coarse-Grained Reconfigurable Arrays

Zhouhua Xie, Kalhan Koul, Jackson Melchert, Priyanka Raina

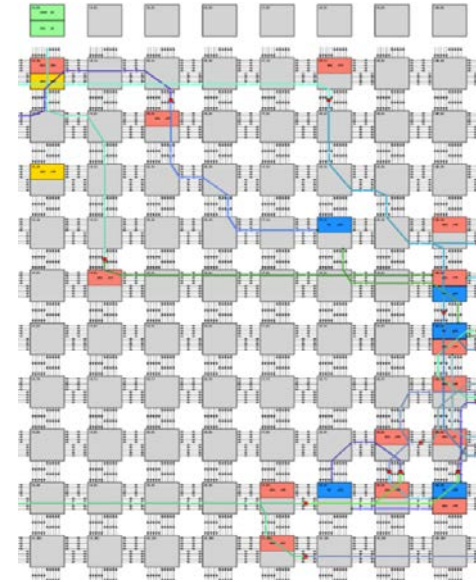
AHA Retreat 2023 Lightning Talk

Current Challenge

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

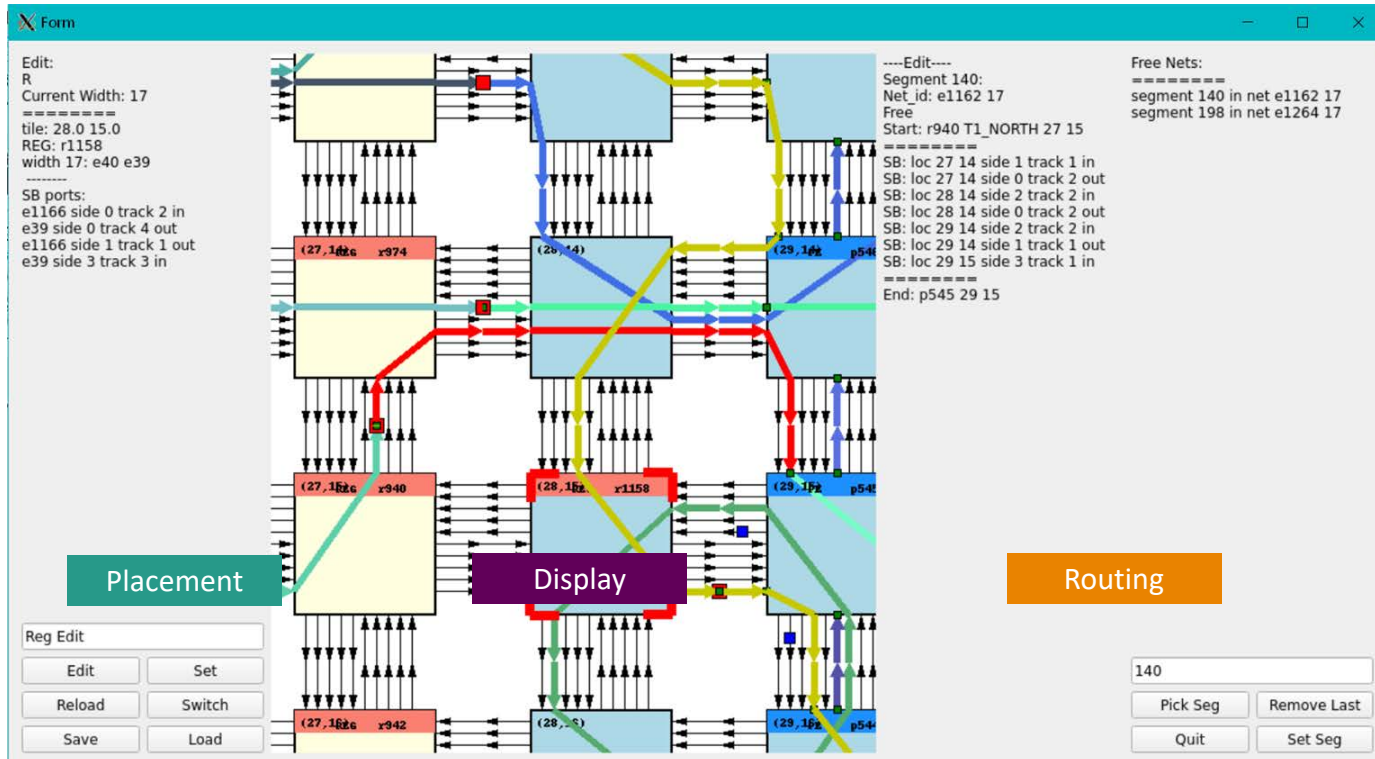
- **Inhomogeneous** resource distribution
- **Limited** resources on placement and routing
- **Complex** design space

Imperfect P&R Algorithm for CGRA



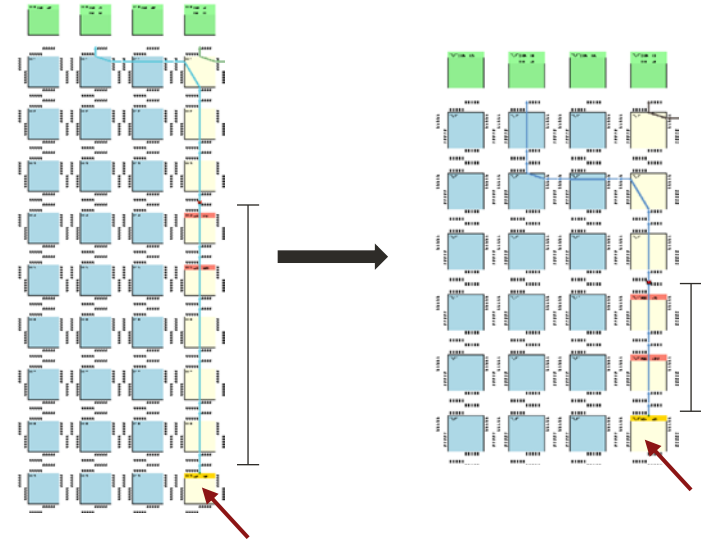
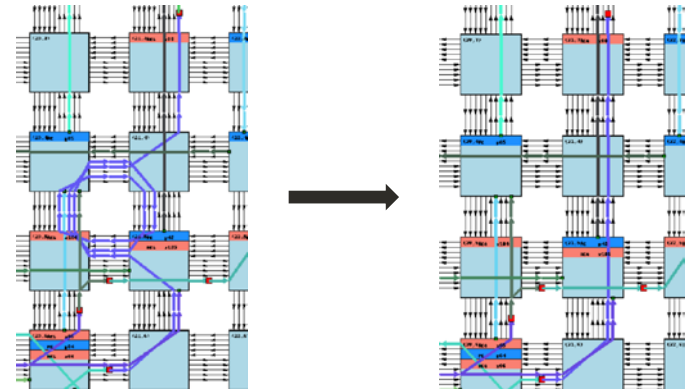
**Problem Example:
Scattered Placement**

Finch



Primary Use Cases

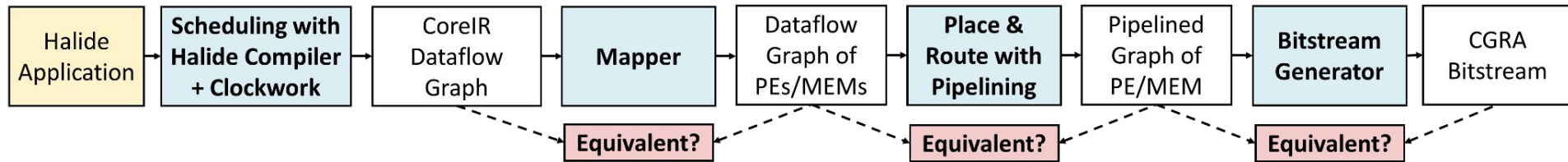
- **Editing for Last Mile Improvement**
- **Design Analysis**
- **Application Design**
- **Prototyping**
- **P&R Algorithm Analysis**



Incorporating Formal Translation Validation into CGRA Compilers

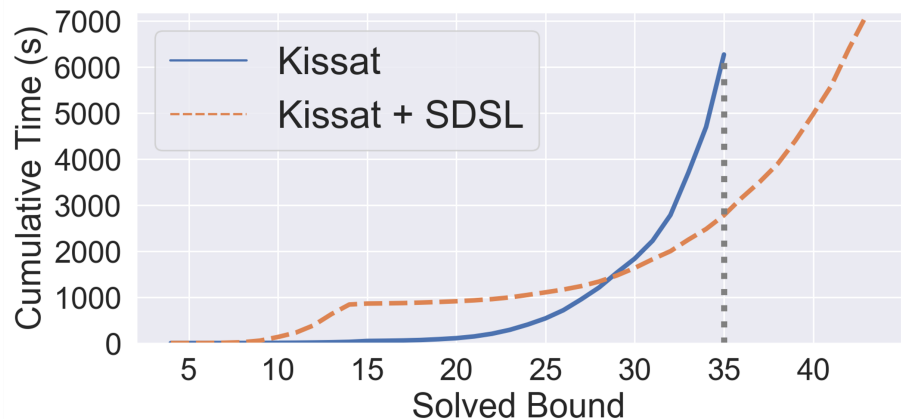
- AHA tools have enabled agile hardware/software codesign
- Verification is still painful and a better solution exists:

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.

Nestan Tsiskaridze¹

¹Stanford University

Clark Barrett¹

Cesare Tinelli²

²The 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

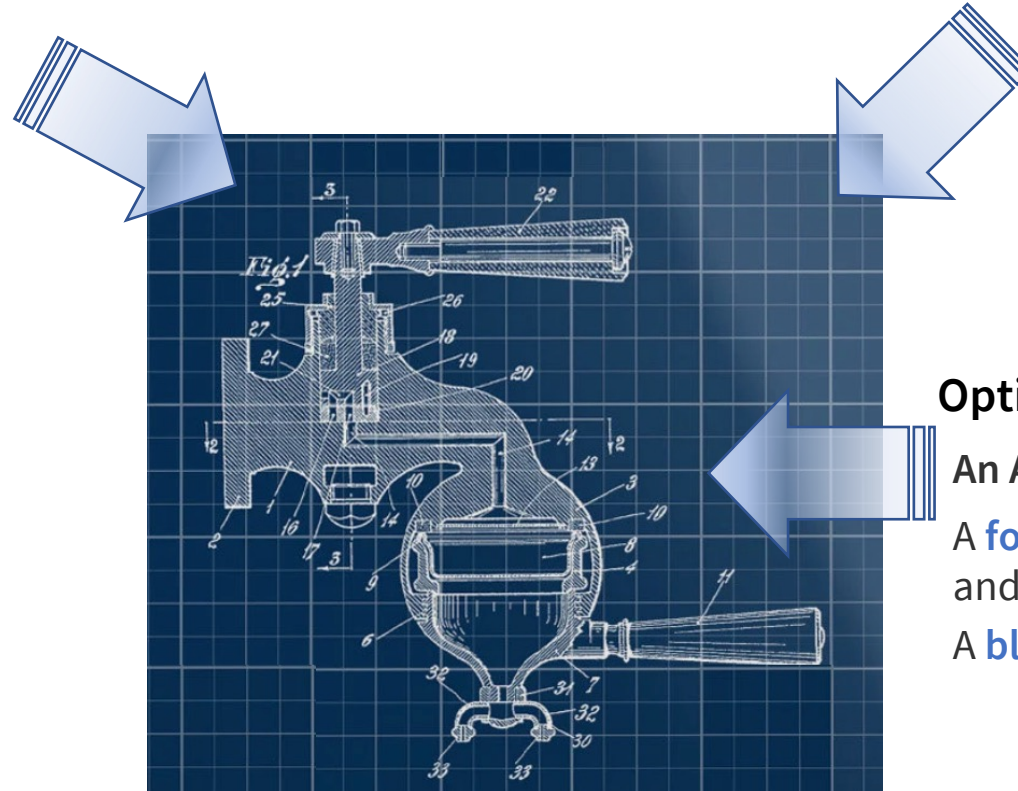
Optimization Modulo Theories (OMT)

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

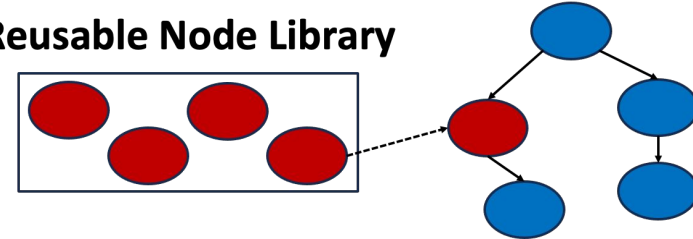
Small-Scale reuse:

- 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

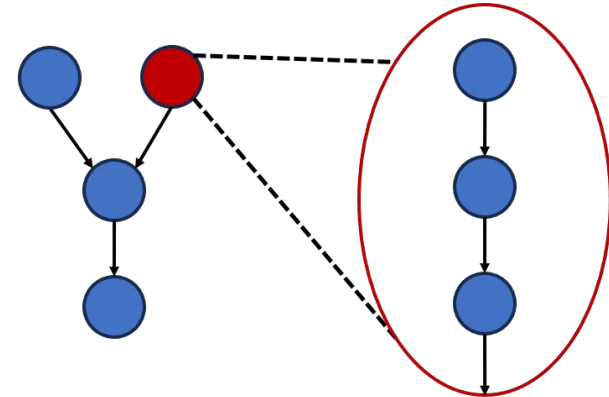
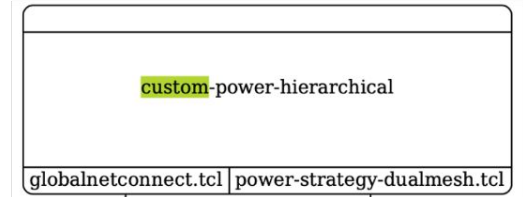
Reusable Node Library



Testing is the Answer

- Augment mflowgen Node primitive with tests
 - Tests can attach to any PD flow graph
- Tests 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!

```
test: <pointer to test graph>
attach_points: PLACE, CTS,
ROUTE
express_mode: skip_opt
```



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



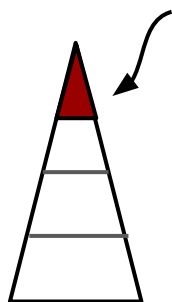
Avanergy: An Architecture-VLSI Abstraction for ~~Automated Energy-Aware Design-Space Exploration Tools~~

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

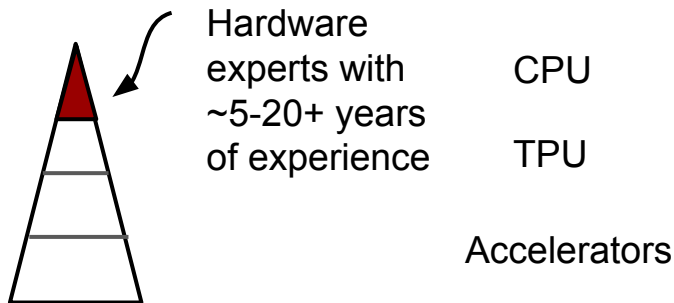
domain enthusiasts with ~1 year or less of experience



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

a very different world of hardware design

1

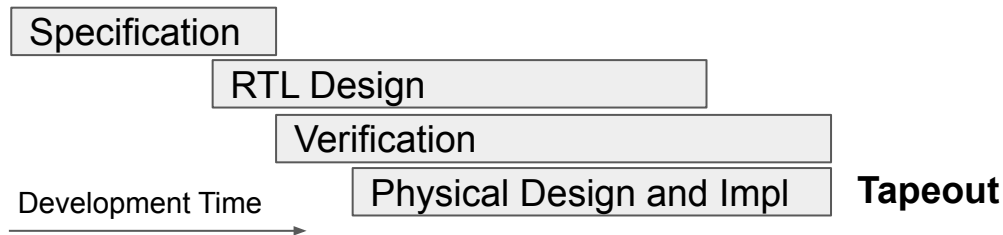


... who are also

domain enthusiasts with ~1 year or less of experience



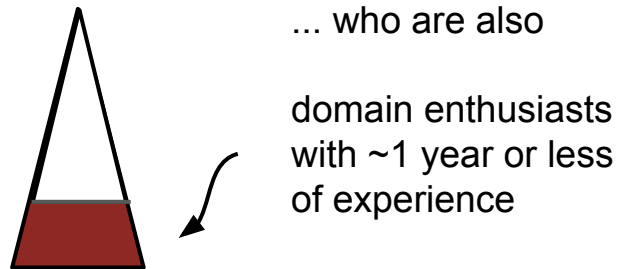
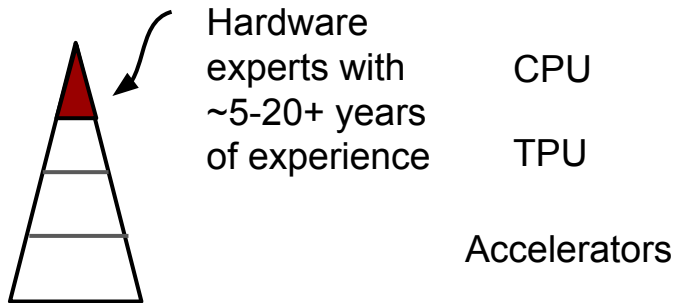
A specific way that things have always been done



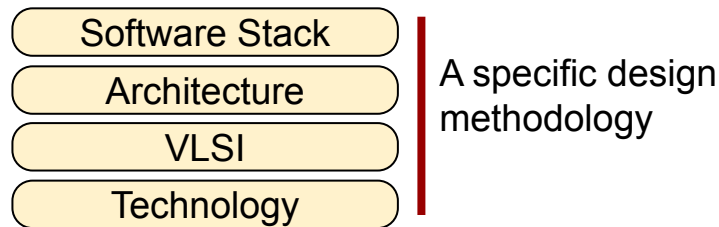
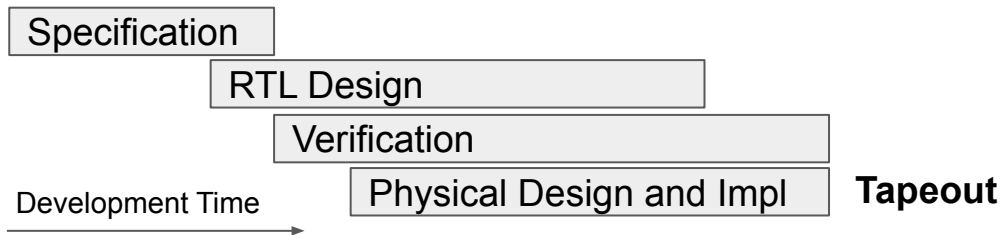
Avanergy: An Architecture-VLSI Abstraction for ~~Automated Energy-Aware Design-Space Exploration Tools~~

a very different world of hardware design

1



A specific way that things have always been done



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

a very different world of hardware design

2



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

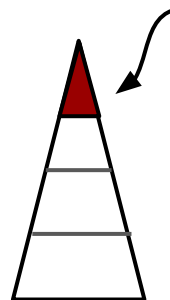
a very different world of hardware design

2



Application domain experts

with ~1 year or less of experience with hardware design



... whose primary jobs are

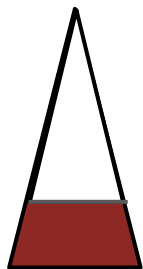
Machine Learning
Image Processing
Video Coding
Cryptography
Wireless

Domain Experts

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

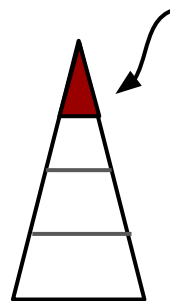
a very different world of hardware design

2



Application domain experts

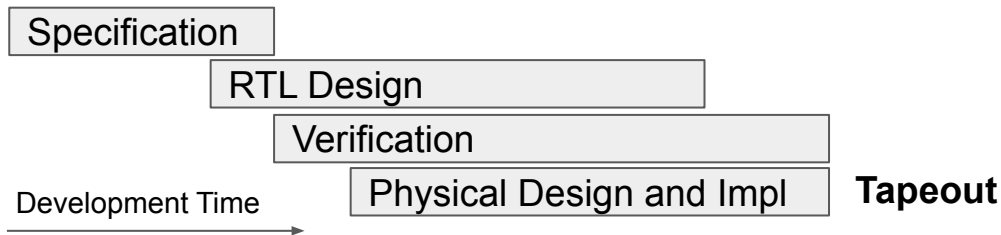
with ~1 year or less of experience with hardware design



... whose primary jobs are

- Machine Learning
- Image Processing
- Video Coding
- Cryptography
- Wireless

Key Question: Do our existing methods work?



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

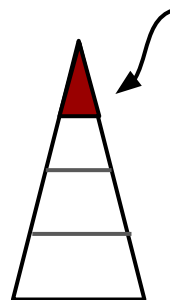
a very different world of hardware design

2



Application domain experts

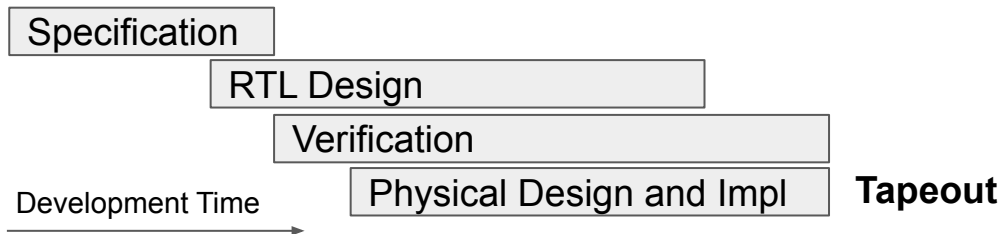
with ~1 year or less of experience with hardware design



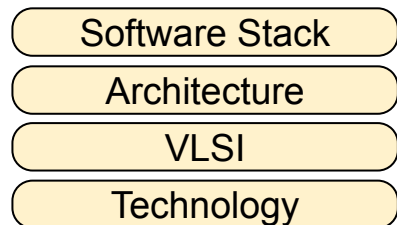
... whose primary jobs are

Machine Learning
Image Processing
Video Coding
Cryptography
Wireless

Key Question: Do our existing methods work?



Domain Experts



Does the stack need to change to "hide" more?

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

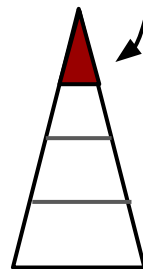
a very different world of hardware design

2



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

Domain Experts

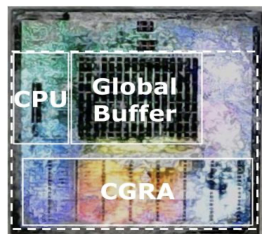
Avanergy is a new **abstraction** that numerically systematizes energy-aware design-space exploration in a way that future automated tools can build upon it

Automatic Discovery of Late Stage Timing Bugs

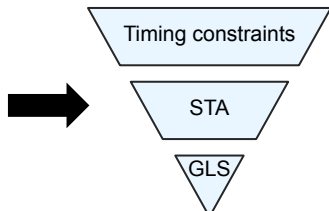


Raj Setaluri, Christopher Torng

Complex chips have timing bugs

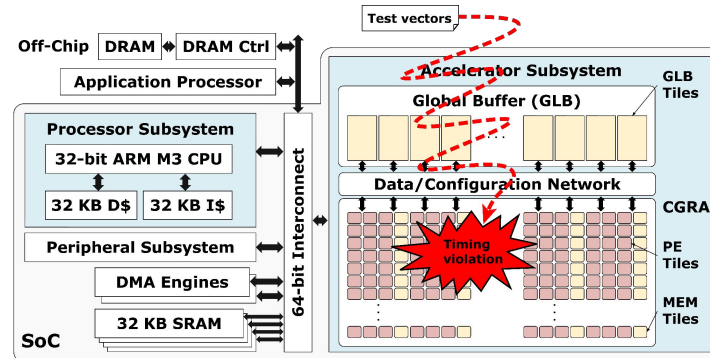


Amber



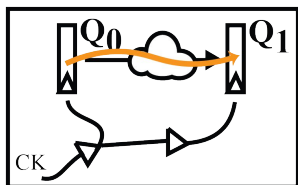
Timing bugs still exist!

Timing bugs are needles in a haystack

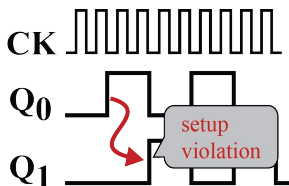


Why do these bugs happen?

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

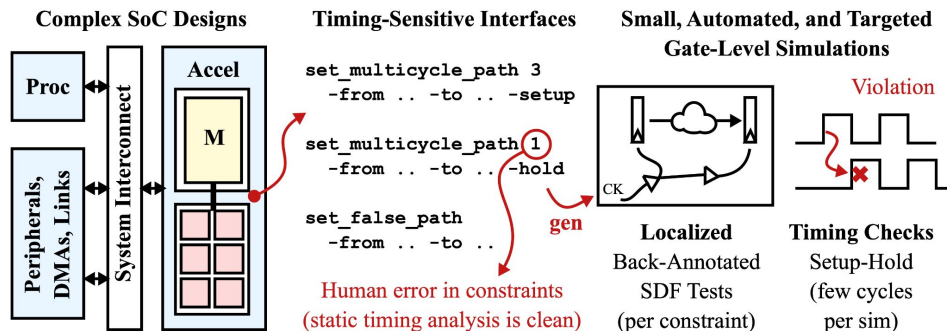


Static timing analysis



Gate-level simulation

Automatically find timing bugs using formal tools



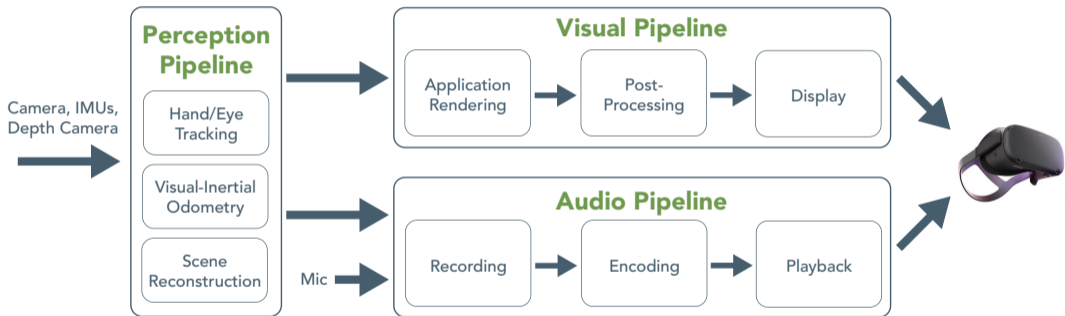
ASPEN: Acceleration of Visual-Inertial Odometry for Extended Reality on an FPGA

Kathleen Feng

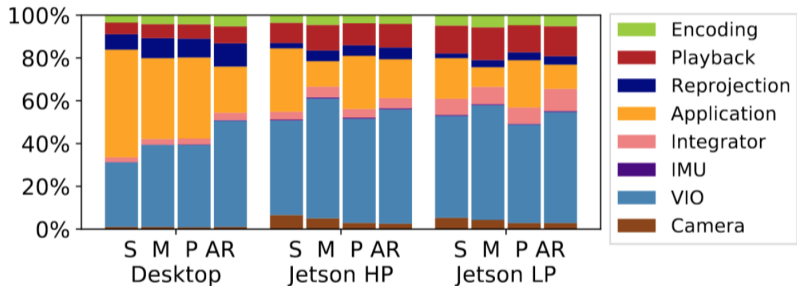
Stanford University

30 August 2023

Extended Reality Pipeline



CPU Performance Breakdown

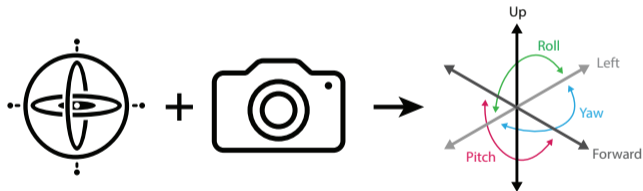


Source: ILLIXR [Huzaiifa 2021]

VIO: Visual Inertial Odometry

Calculates 3D user position from sensors

- IMUs, cameras $\Rightarrow (x, y, z, \theta, \phi, \psi)$
- Most dominating subtask, represents $\sim 40\%$ of XR workload



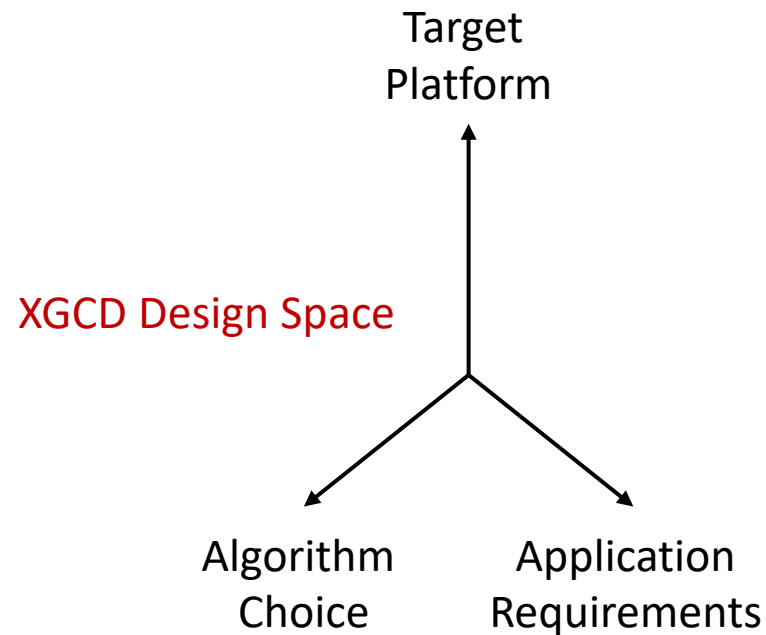
Using OpenVINS as gold model [<https://docs.openvins.com/index.html>]

A Fast Large-Integer Extended GCD Algorithm and Hardware Design

Kavya Sreedhar, Mark Horowitz, Christopher Torng

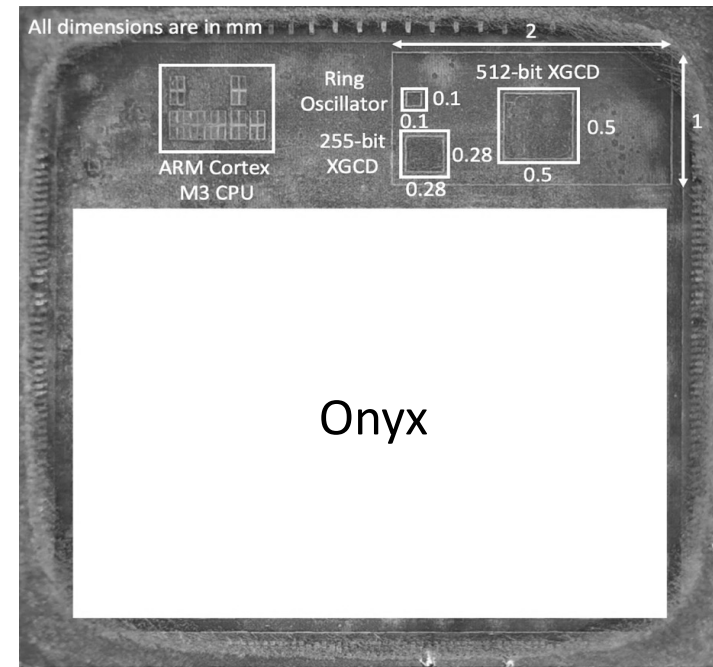
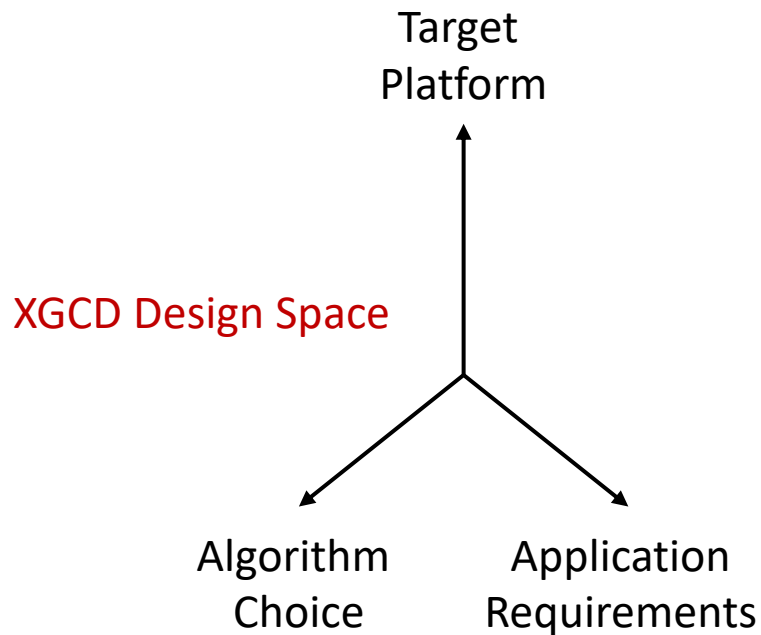
A Fast Large-Integer Extended GCD Algorithm and Hardware Design

Kavya Sreedhar, Mark Horowitz, Christopher Torng



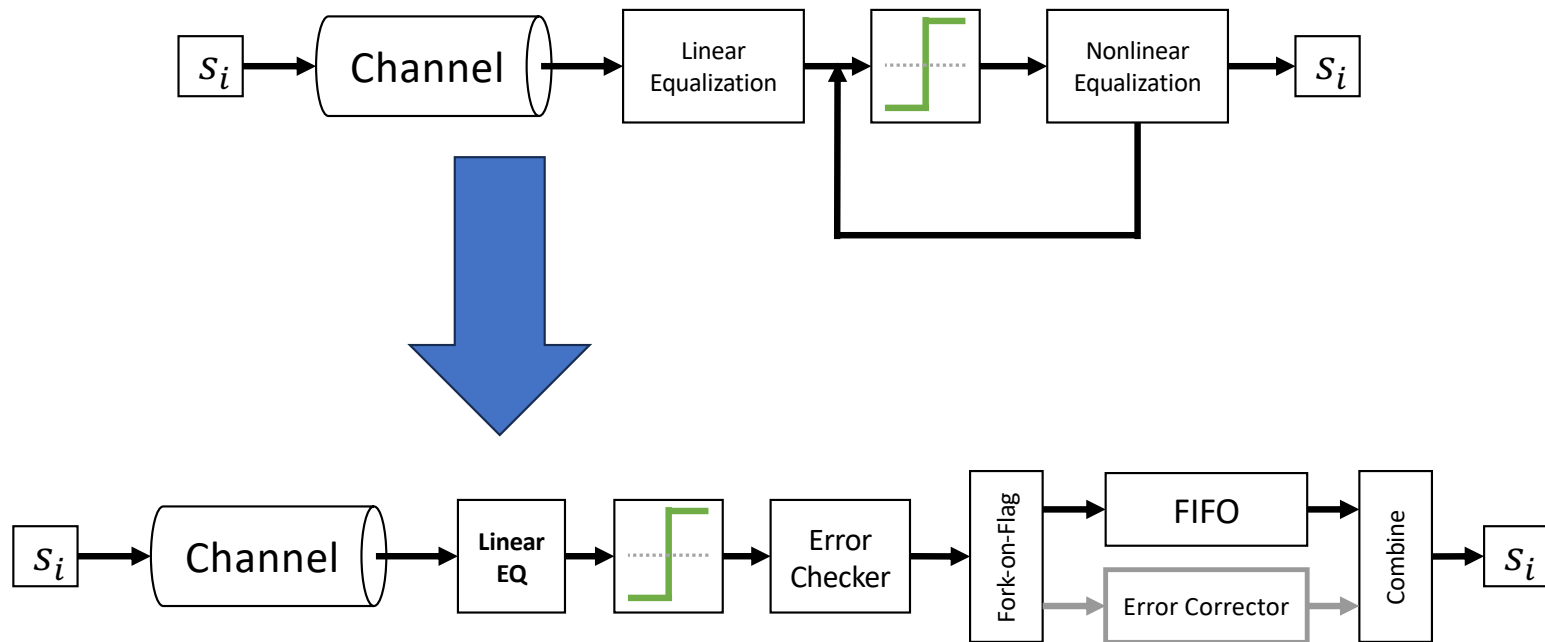
A Fast Large-Integer Extended GCD Algorithm and Hardware Design

Kavya Sreedhar, Mark Horowitz, Christopher Torng



Interrupt-driven MLSD-based Links

Zach Myers, Stanford VLSI Group



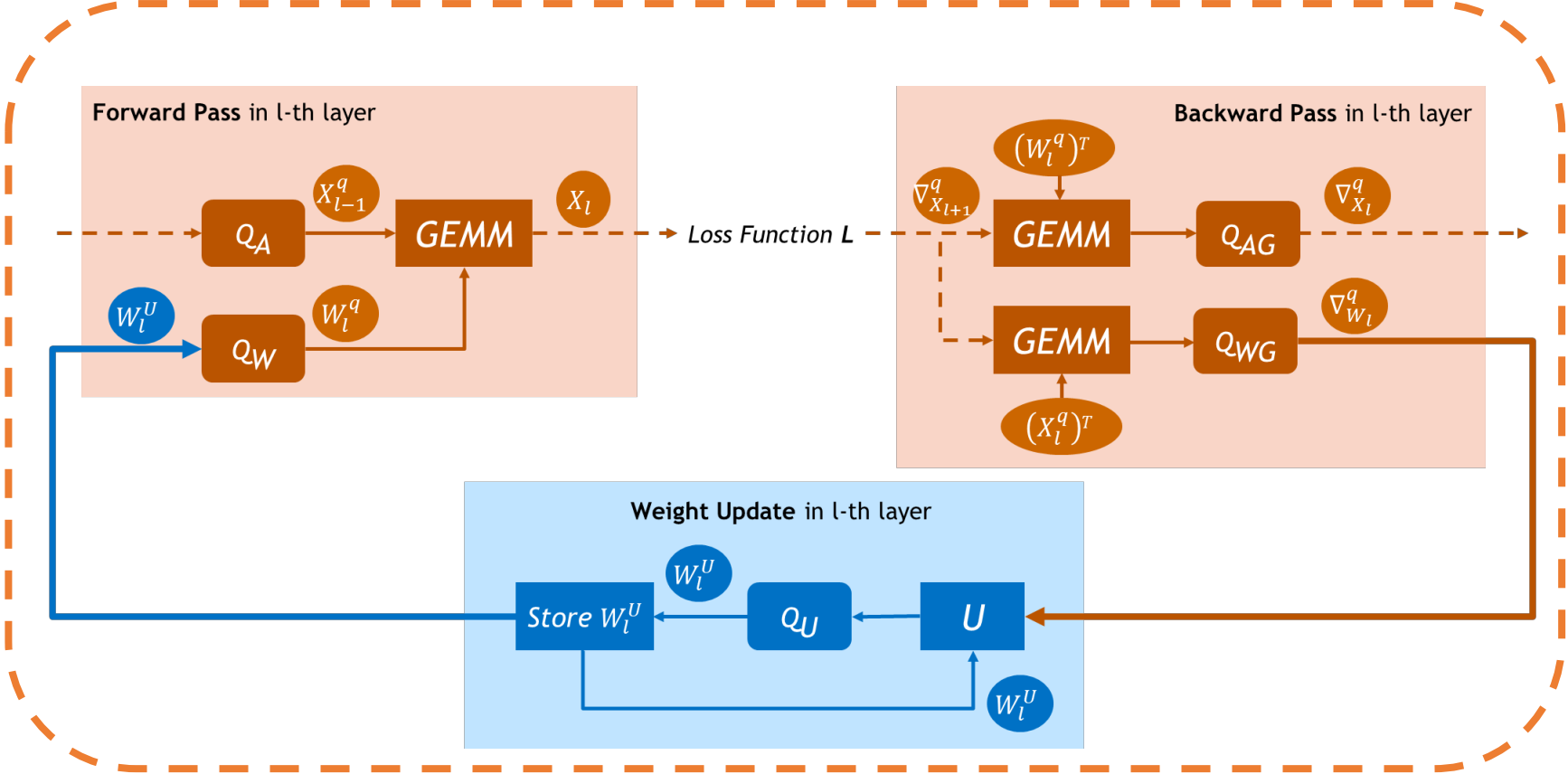
Co-Designing AI Models and Embedded DRAMs for Efficient On-Device ML Training

Thierry Tambe



Stanford
University

Data transience is pervasive



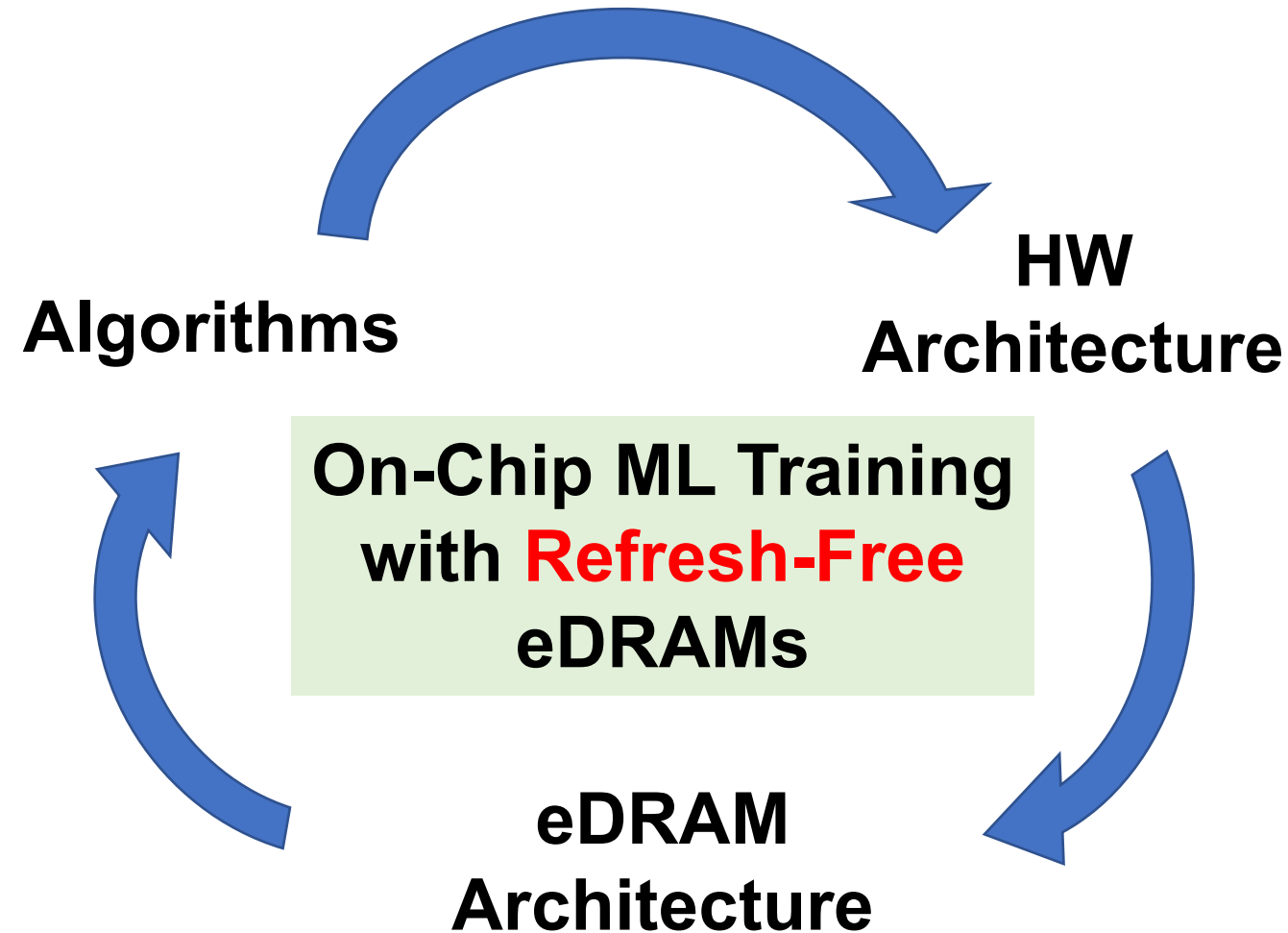
Computational graph of DL training

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

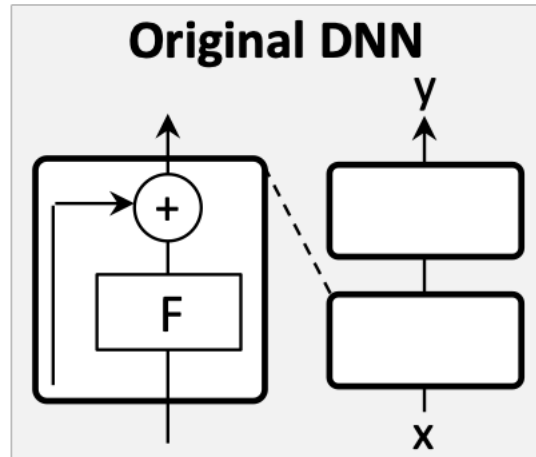
eDRAM as main on-chip storage medium

- + 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

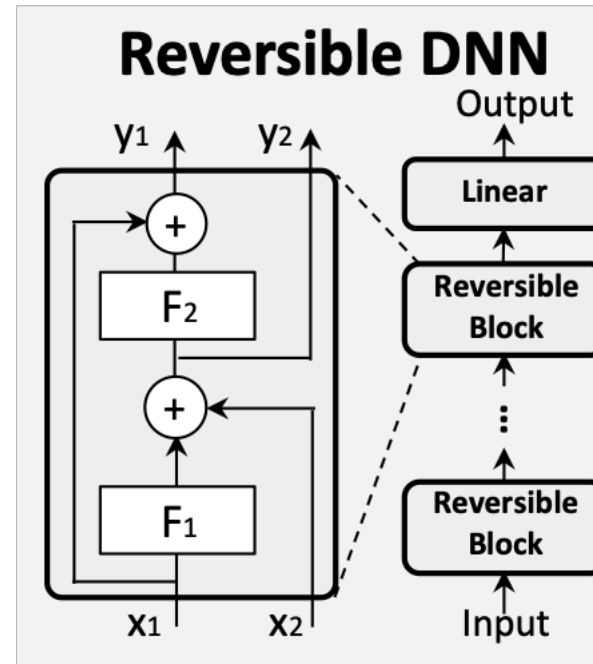
Algorithm-System Co-Design for eDRAM-based Computing



Reversible neural networks to avoid buffering activations



Convert



Equations during the training process

Forward pass:

$$y_2 = F_1(x_1) + x_2$$

$$y_1 = F_2(y_2) + x_1$$

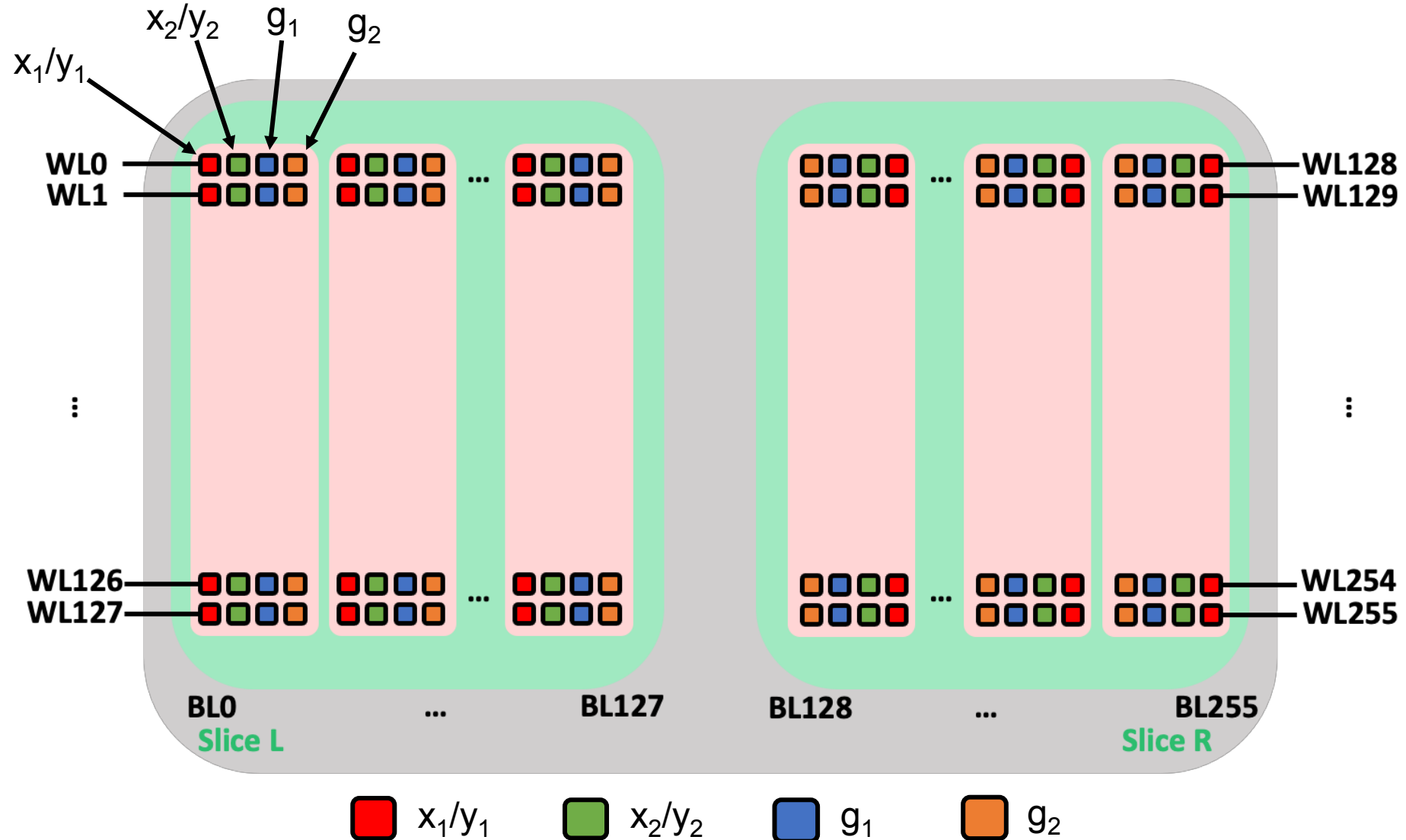
Backward Pass:

$$x_1 = y_1 - F_2(y_2)$$

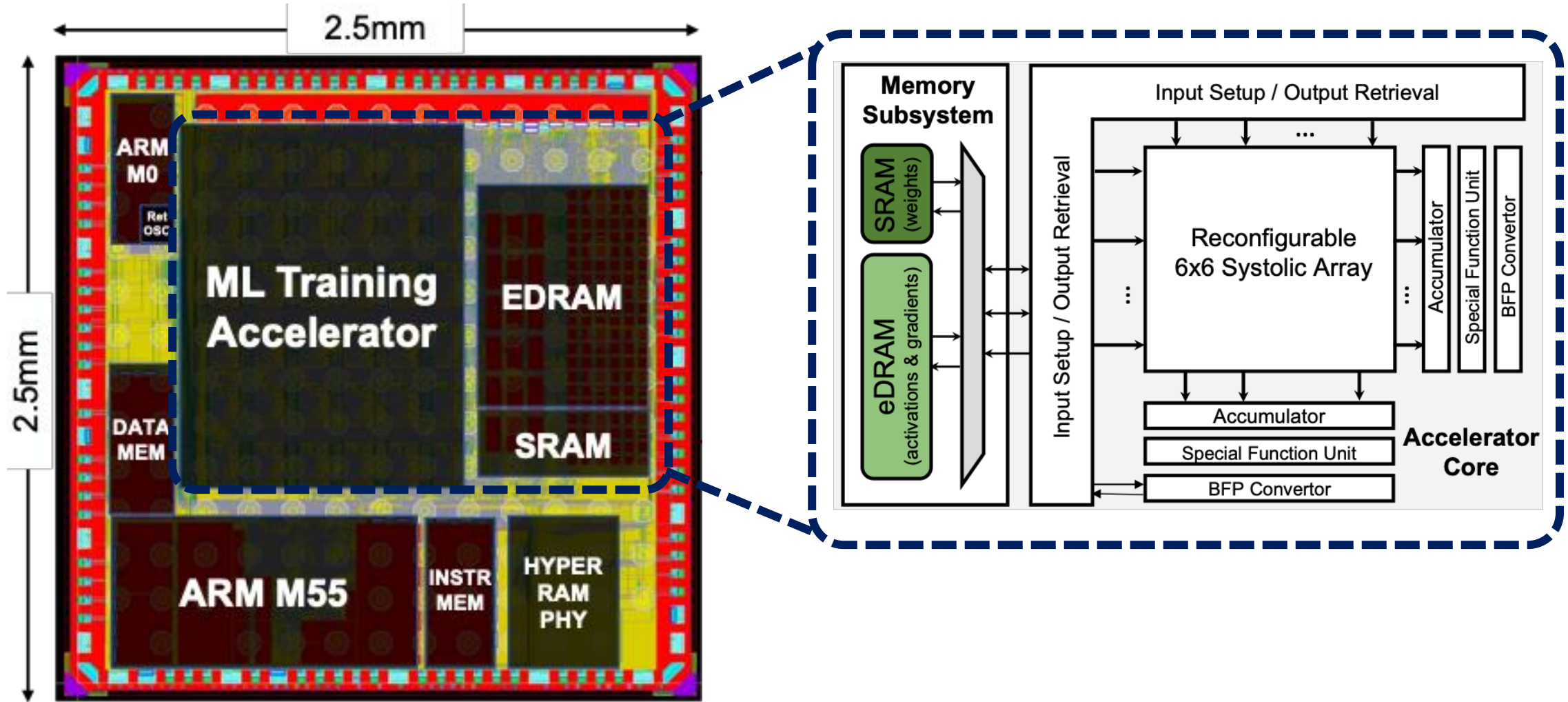
$$x_2 = y_2 - F_1(x_1)$$

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!