Onyx: A Programmable Accelerator for Sparse Tensor Algebra

Kalhan Koul[1], Maxwell Strange[1], Jackson Melchert[1], Alex Carsello[1], Yuchen Mei[1], Olivia Hsu[1], Taeyoung Kong[1], Po-Han Chen[1], Huifeng Ke[1], Keyi Zhang[1], Qiaoyi Liu[1], Gedeon Nyengele[1], Akhilesh Balasingam[1], Jayashree Adivarahan[1], Ritvik Sharma[1], Zhouhua Xie[1], Christopher Torng[2], Joel Emer[3], Fredrik Kjolstad[1], Mark Horowitz[1], Priyanka Raina[1]

[1]Stanford University, CA, USA; [2]University of Southern California, CA, USA; [3]MIT, MA, USA

Stanford University

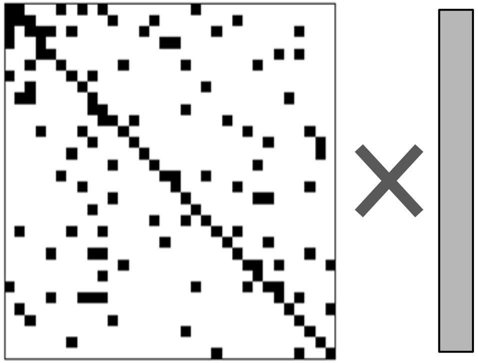
Sparse Applications

- Applications ranging from scientific computing to machine learning can have **extremely** sparse inputs

Sparse Applications

- Applications ranging from scientific computing to machine learning can have **extremely** sparse inputs

Scientific Computing

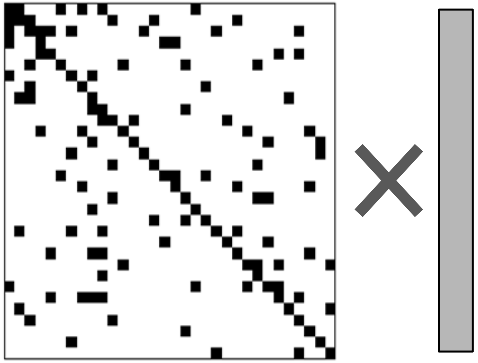


Up to 99.9% sparse matrices

Sparse Applications

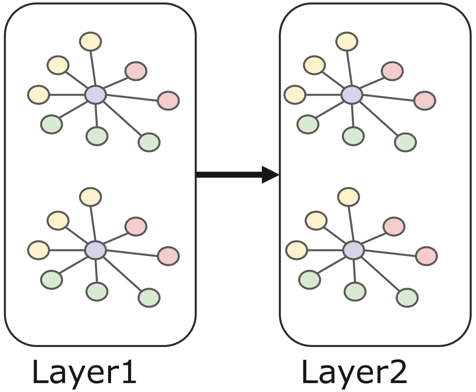
- Applications ranging from scientific computing to machine learning can have **extremely** sparse inputs

Scientific Computing



Up to 99.9% sparse matrices

Graph Neural Networks

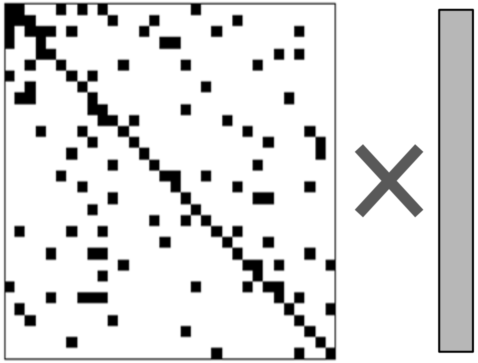


Up to 99.9% sparse matrices

Sparse Applications

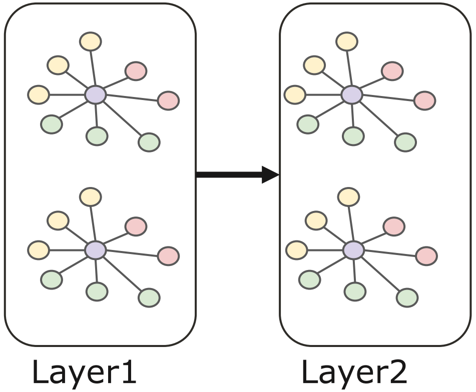
- Applications ranging from scientific computing to machine learning can have **extremely** sparse inputs

Scientific Computing



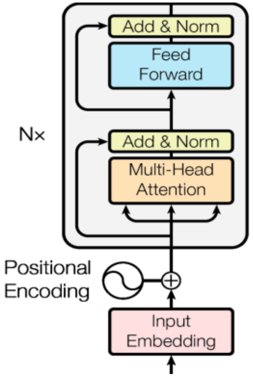
Up to 99.9% sparse matrices

Graph Neural Networks



Up to 99.9% sparse matrices

Sparse Transformers

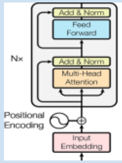


Up to 93% sparse matrices

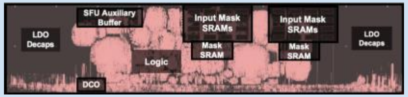
End-to-End Sparse Accelerators

- Hardware accelerators for end-to-end applications exploiting sparsity are fixed function and become obsolete as applications evolve

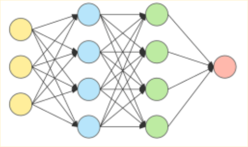
Transformer Inference



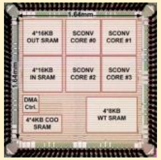
STP
[ISSCC 2023]



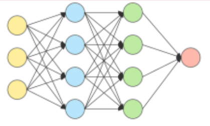
2D/3D CNN for Point Clouds

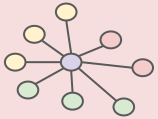


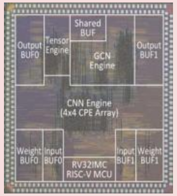
SCONV
[ISSCC 2023]



CNN + Graph Conv Network

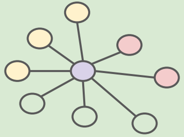




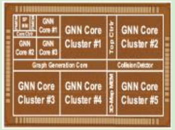


AR SoC
[VLSI 2022]

3D Navigation



GPPU
[VLSI 2023]

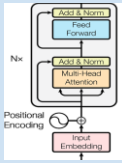


Stanford University

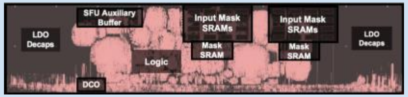
End-to-End Sparse Accelerators

- Hardware accelerators for end-to-end applications exploiting sparsity are fixed function and become obsolete as applications evolve

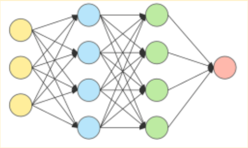
Transformer Inference



STP
[ISSCC 2023]

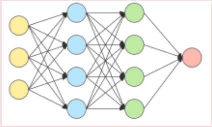


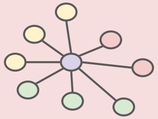
2D/3D CNN for Point Clouds

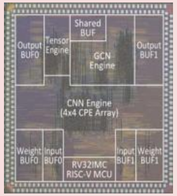


SCONV
[ISSCC 2023]

CNN + Graph Conv Network

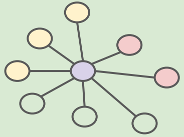




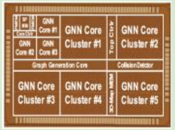


AR SoC
[VLSI 2022]

3D Navigation



GPPU
[VLSI 2023]



✅ Energy and area efficient

❌ Accelerate a fixed type of application

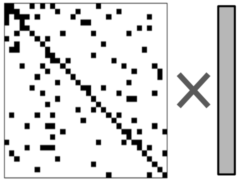
❌ Do not adapt to evolving application domains

Stanford University

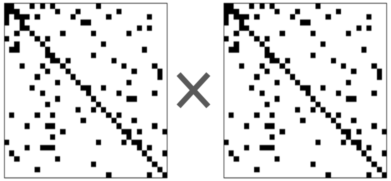
Kernel-Level Sparse Accelerators

- Sparse kernel accelerators focus only on sparse matrix-matrix or matrix-vector multiplication

Supported



matrix times vector

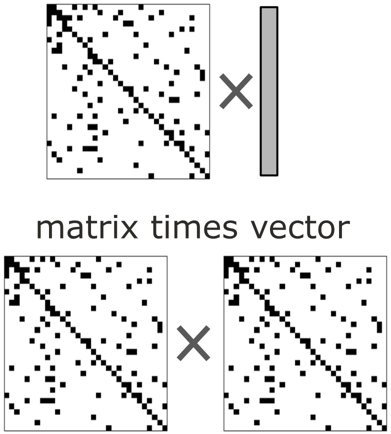


matrix times matrix

Kernel-Level Sparse Accelerators

- Sparse kernel accelerators focus only on sparse matrix-matrix or matrix-vector multiplication
- These accelerators leave out support for high-dimensional tensors, multi-input complex expressions and fusion

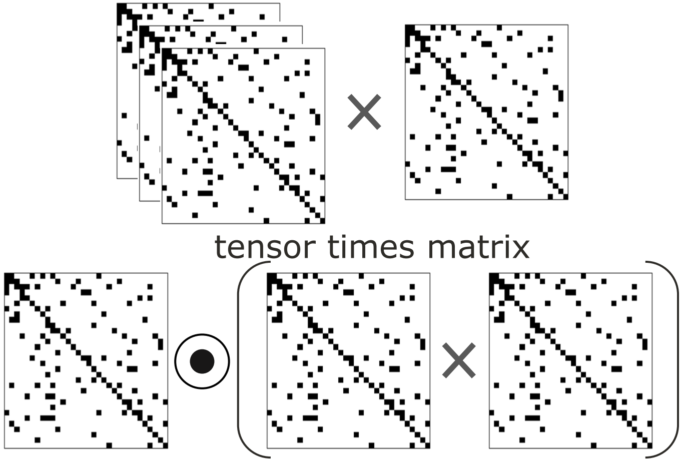
Supported



matrix times vector

matrix times matrix

Unsupported

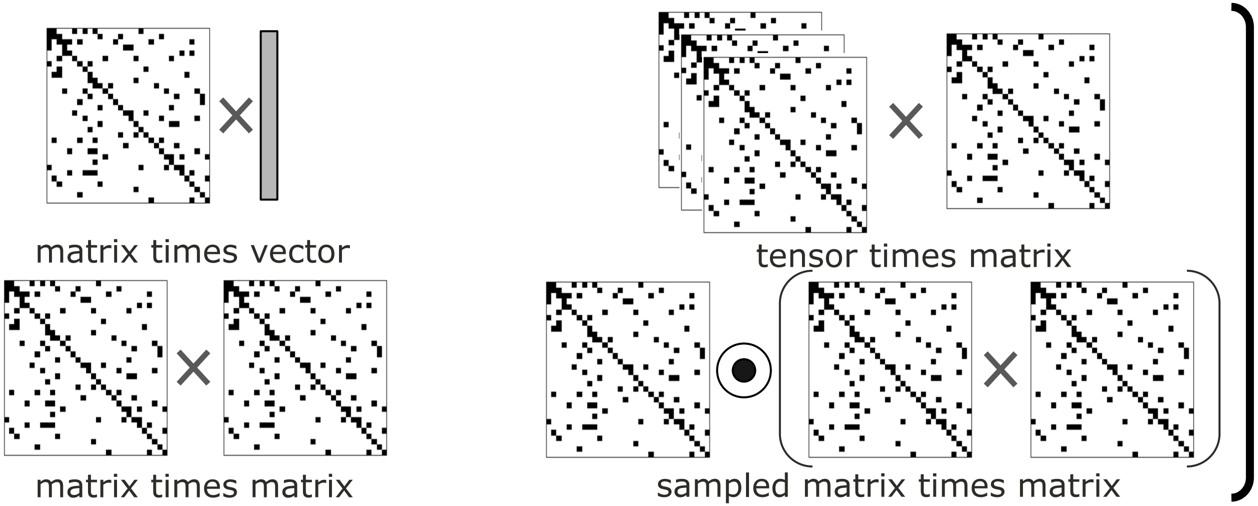


tensor times matrix

sampled matrix times matrix

Kernel-Level Sparse Accelerators

- Sparse kernel accelerators focus only on sparse matrix-matrix or matrix-vector multiplication
- These accelerators leave out support for high-dimensional tensors, multi-input complex expressions and fusion



Supported

matrix times vector

matrix times matrix

Unsupported

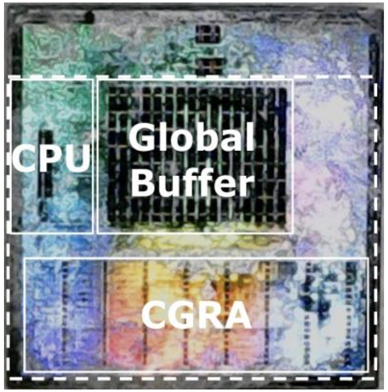
tensor times matrix

sampled matrix times matrix

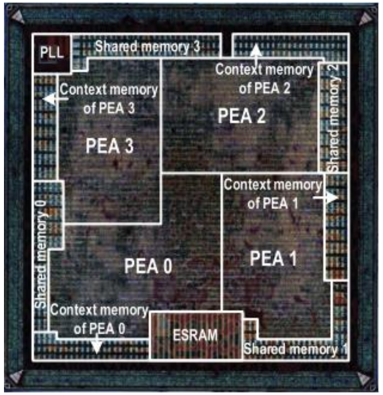
Require a highly programmable accelerator!

Programmable Accelerators

- But most programmable accelerators **only** focus on dense applications



Amber CGRA
[VLSI 2022]



MIMO CGRA
[JSSC 2020]

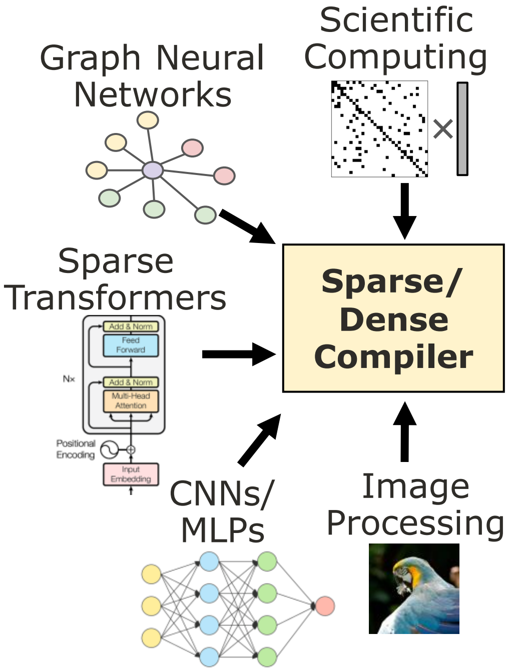
✅ Energy and area efficiency approaching ASICs

✅ Accelerate a domain of applications

✅ Adaptable to evolving dense application domains

Onyx

- Programmable accelerator for any tensor algebra expression



Graph Neural Networks

Scientific Computing

Sparse Transformers

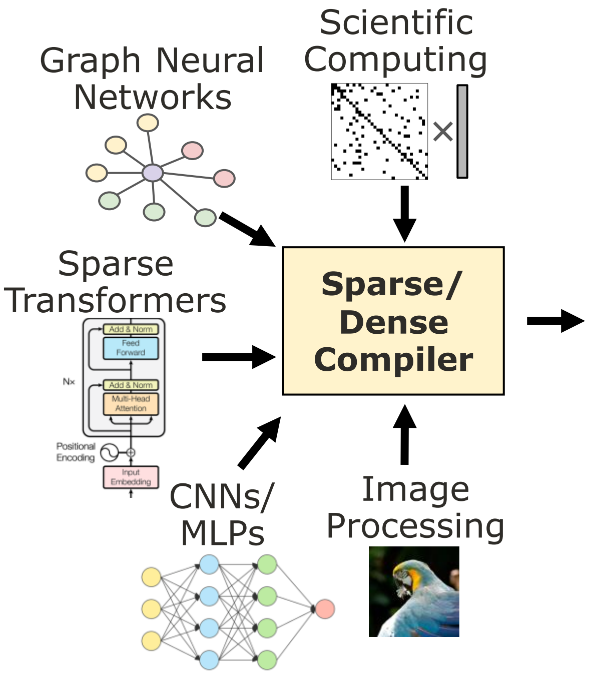
Sparse/ Dense Compiler

CNNs/ MLPs

Image Processing

Onyx

- Programmable accelerator for <u>any</u> tensor algebra expression



Graph Neural Networks

Scientific Computing

Sparse Transformers

Sparse/ Dense Compiler

CNNs/ MLPs

Image Processing

$$A = Bc + a \qquad a = Bc$$

$$A = B \odot C \qquad A = B + C \qquad a = \alpha Bc + \beta a$$

$$A = BCd \qquad A = \alpha B \qquad A = 0 \qquad A = BC$$

$$a = b \odot c \qquad A = B \odot (CD)$$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A = B^T \qquad a = B^T Bc$$

$$A_{ik} = \sum_{j} B_{ijk} c_j \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ij} = (\sum_{k} B_{ijk} C_{ijk}) + D_{ij}$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}} \qquad \tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

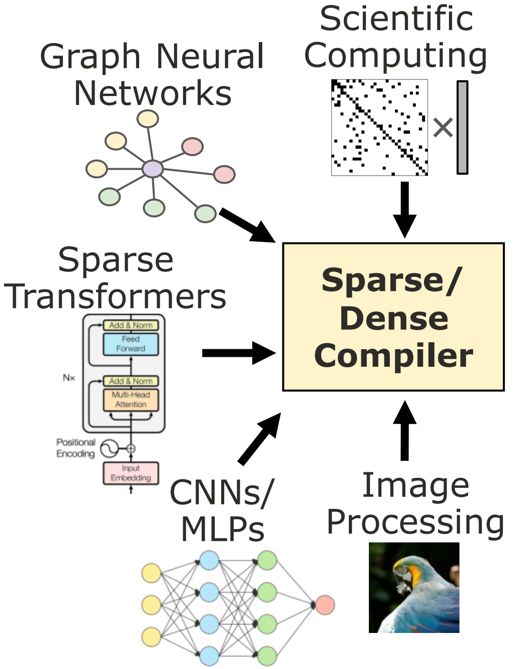
$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

Arbitrary sparse/ dense tensor algebra expressions

Sparse abstract machine (SAM)/CoreIR dataflow graph

Onyx

- Programmable accelerator for <u>any</u> tensor algebra expression



Graph Neural Networks

Scientific Computing

Sparse Transformers

Sparse/ Dense Compiler

CNNs/ MLPs

Image Processing

$$A = Bc + a \qquad a = Bc$$

$$A = B \odot C \qquad A = B + C \qquad a = \alpha Bc + \beta a$$

$$A = BCd \qquad A = \alpha B \qquad A = 0 \qquad A = BC$$

$$a = b \odot c \qquad A = B \odot (CD)$$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A = B^T \qquad a = B^T Bc$$

$$A_{ik} = \sum_{j} B_{ijk} c_j \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

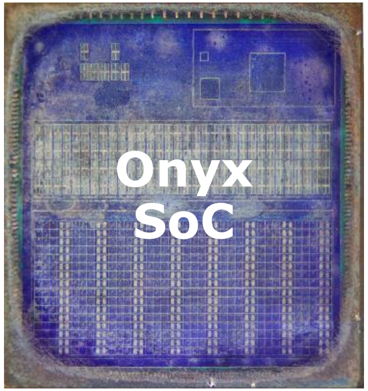
$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ij} = (\sum_{k} B_{ijk} C_{ijk}) + D_{ij}$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}} \qquad \tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

Arbitrary sparse/ dense tensor algebra expressions

Sparse abstract machine (SAM)/CoreIR dataflow graph



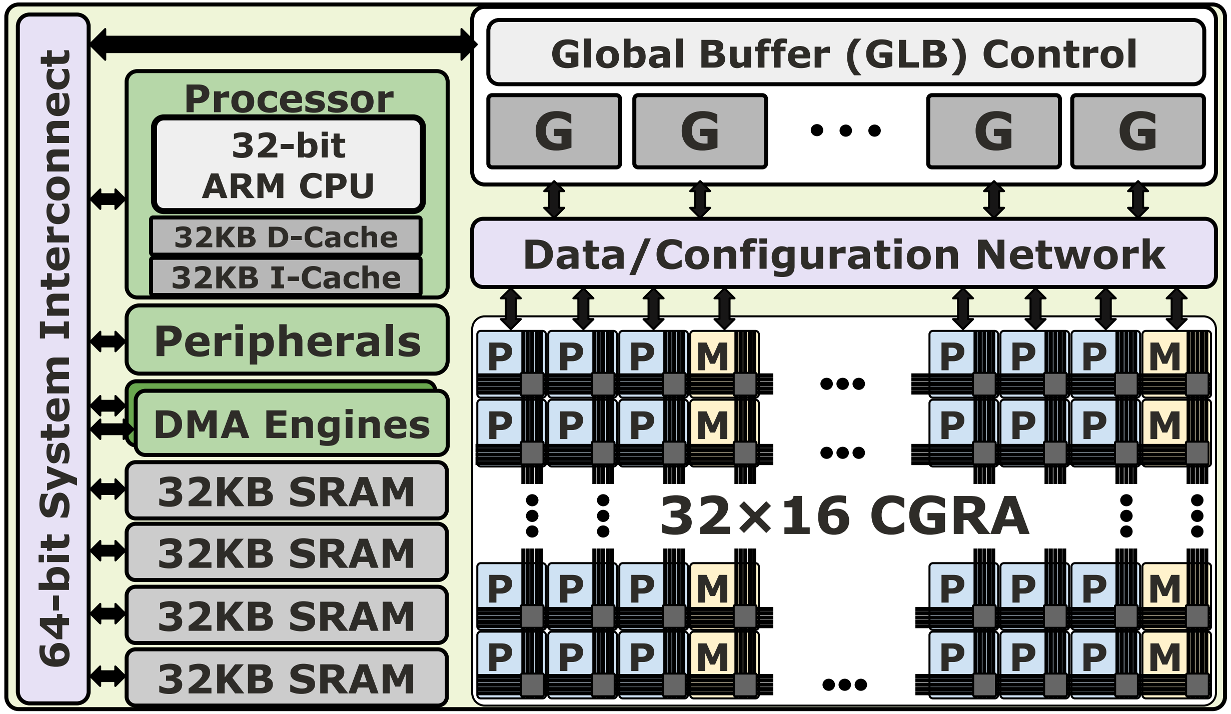
✓ Energy and area efficient

✓ Accelerates an application domain

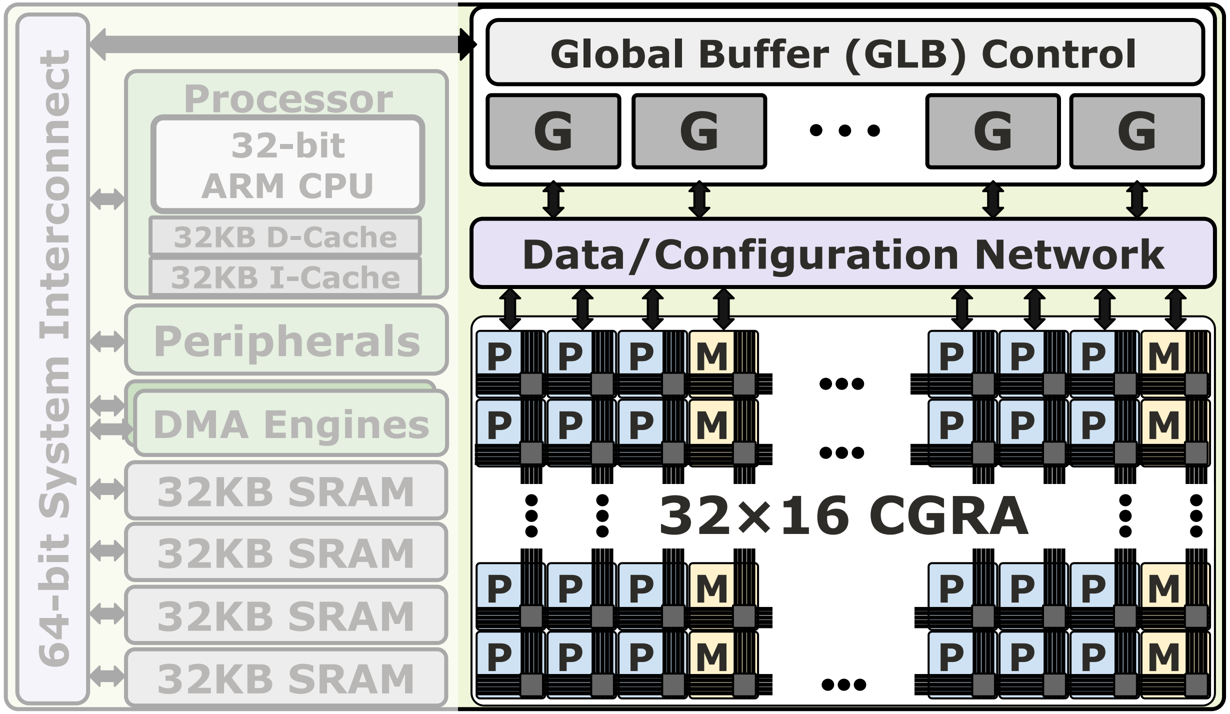
✓ Configurable for sparse + dense

✓ Adaptable to fast evolving domains

Onyx Architecture

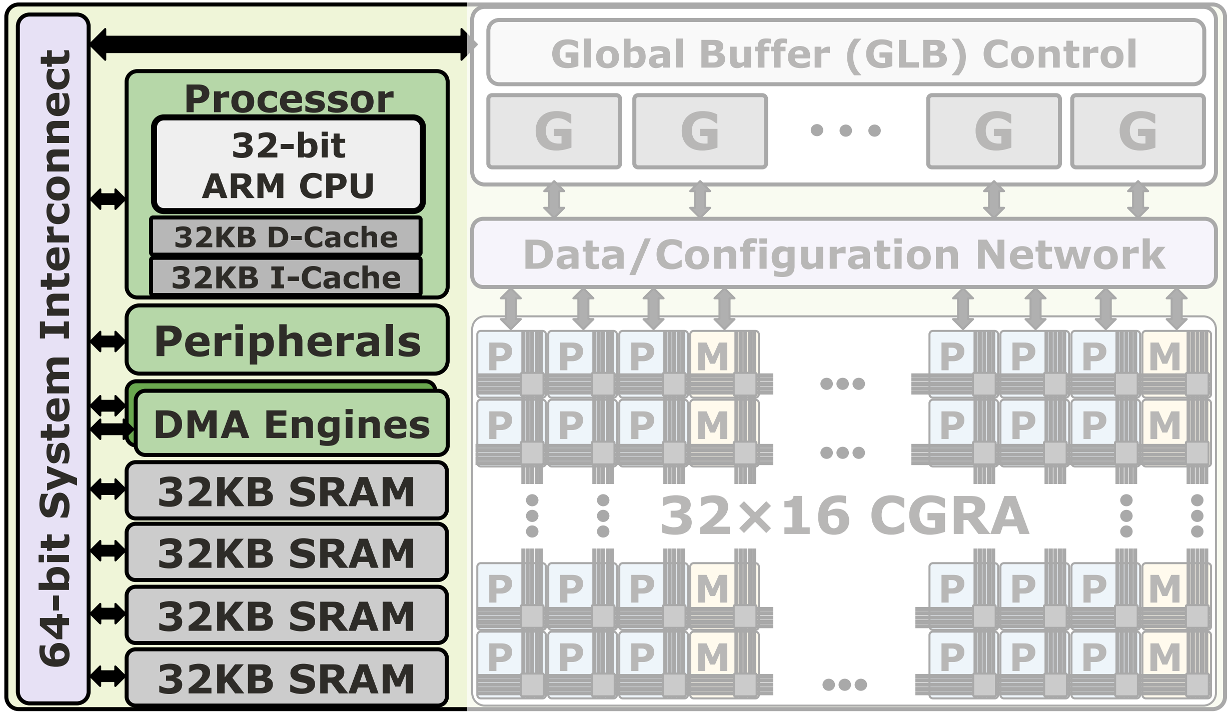


Onyx Architecture



G	16 256 KB Global Buffer Tiles (GLB Tile)
P	384 INT16/BFloat16 Processing Element Tiles (PE Tile)
M	128 4 KB Memory Tiles (MEM Tile)
▦	Switch Box & Routing Tracks

Onyx Architecture

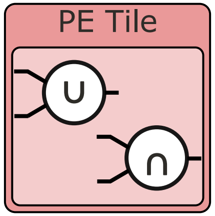


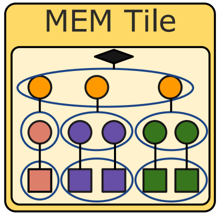
Contributions

Contributions

Sparse Acceleration Hardware

Composable primitives for accelerating arbitrary sparse tensor algebra kernels

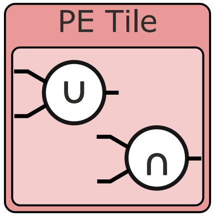


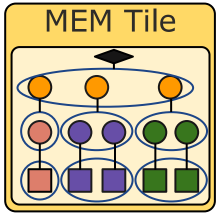


Contributions

Sparse Acceleration Hardware

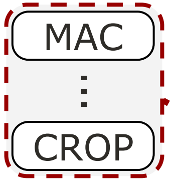
Composable primitives for accelerating arbitrary sparse tensor algebra kernels



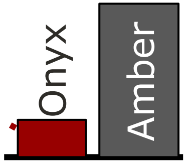


Dense Acceleration Improvements

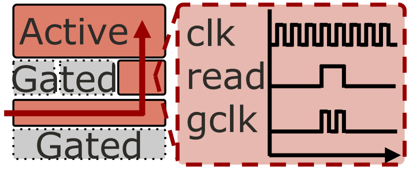
Compute and memory controller optimizations for dense applications



New PE OPs



PE Usage per App

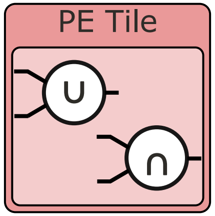


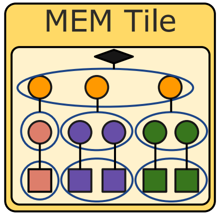
Dynamic-/Static- GLB Clock Gating

Contributions

Sparse Acceleration Hardware

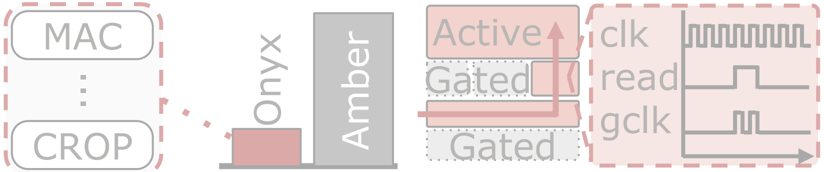
Composable primitives for accelerating arbitrary sparse tensor algebra kernels





Dense Acceleration Improvements

Compute and memory controller optimizations for dense applications



New PE OPs

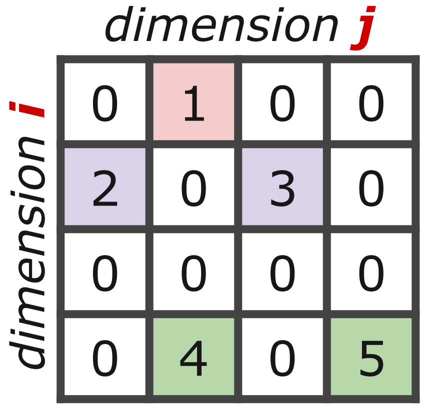
PE Usage per App

Dynamic-/Static-GLB Clock Gating

Sparse Tensor Abstraction

- Sparse tensors can be represented as fibertree [1] structures

Tensor



0	1	0	0
2	0	3	0
0	0	0	0
0	4	0	5

[1] V. Sze et al., M&C’2020.

Sparse Tensor Abstraction

- Sparse tensors can be represented as fibertree [1] structures

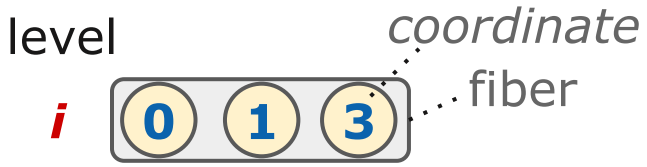
Tensor

dimension j

dimension i

$$\begin{array}{c|cccc} 0 & 0 & 1 & 0 & 0 \\ 1 & 2 & 0 & 3 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 3 & 0 & 4 & 0 & 5 \end{array}$$

Fibertree



[1] V. Sze et al., M&C’2020.

Sparse Tensor Abstraction

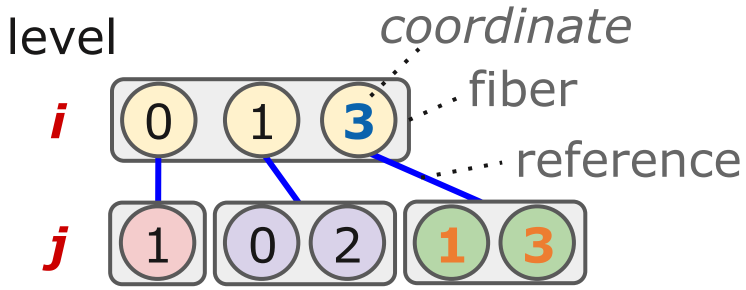
- Sparse tensors can be represented as fibertree [1] structures

Tensor

dimension ***j***

dimension ***i***	0	1	2	3
0	0	1	0	0
1	2	0	3	0
2	0	0	0	0
3	0	4	0	5

Fibertree

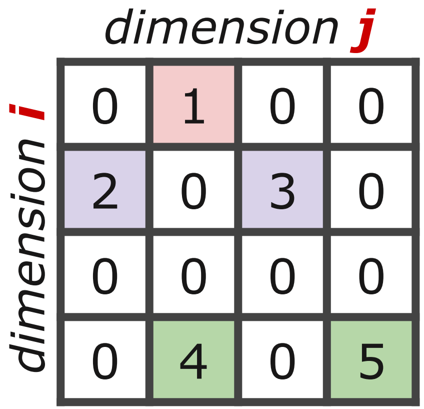


[1] V. Sze et al., M&C'2020.

Sparse Tensor Abstraction

- Sparse tensors can be represented as fibertree [1] structures

Tensor

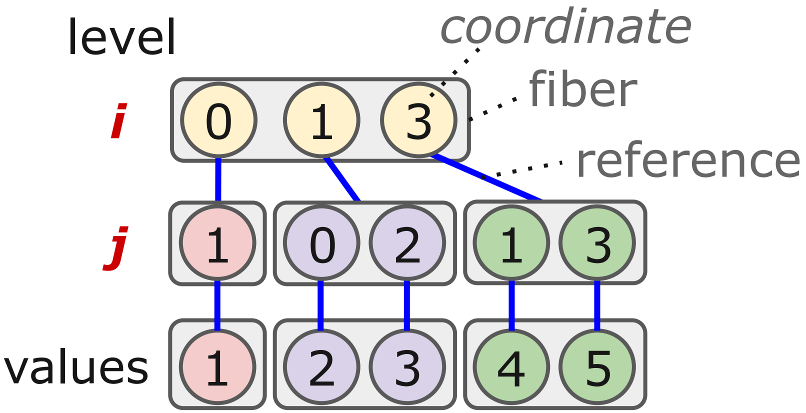


dimension j

dimension i

0	1	0	0
2	0	3	0
0	0	0	0
0	4	0	5

Fibertree

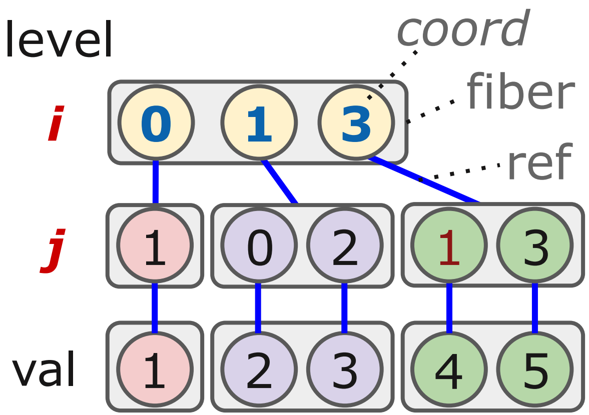


[1] V. Sze et al., M&C’2020.

Stream and Storage Format

- Fibertree in stream format

Fibertree



level

coord

fiber

ref

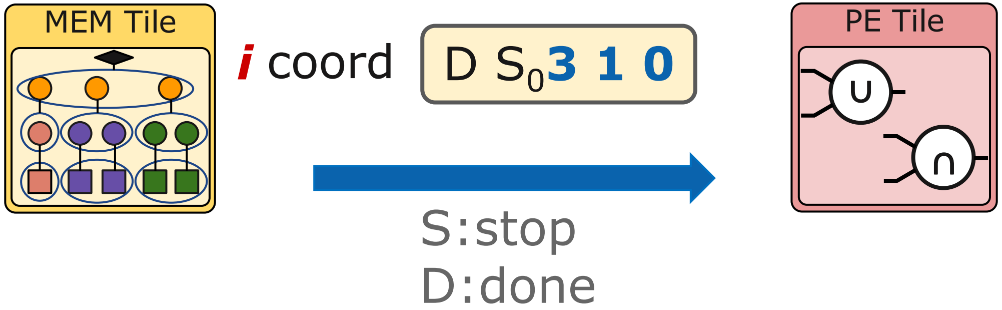
i 0 1 3

j 1 0 2 1 3

val 1 2 3 4 5

coordinates: coord
references: ref
segments: seg   *values: val*

Stream Format

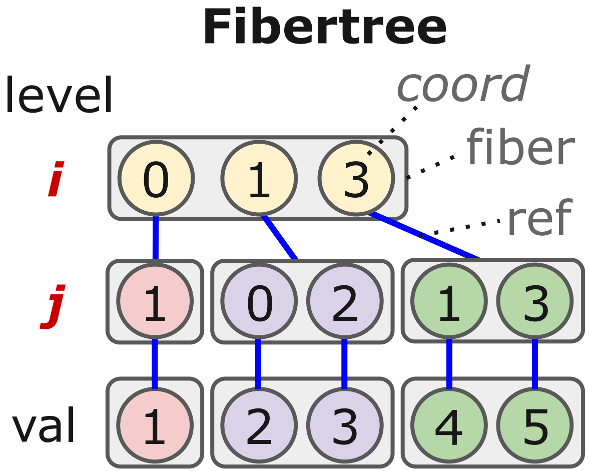


i coord D S_0 **3 1 0**

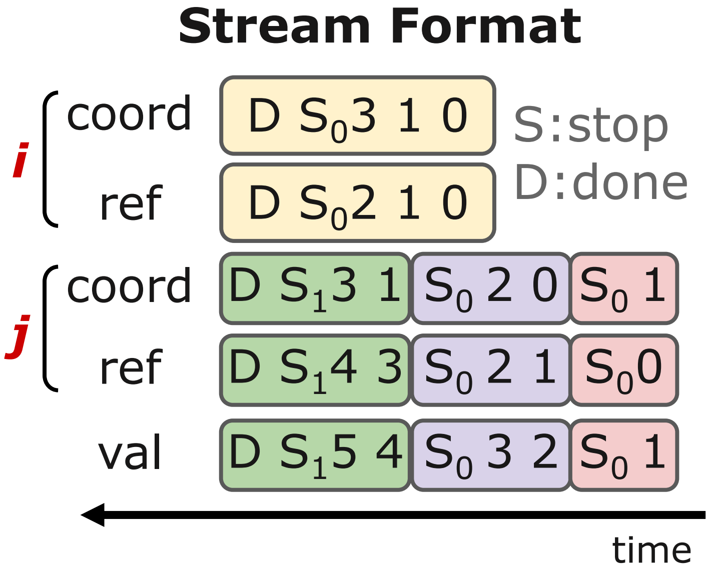
S:stop
D:done

Stream and Storage Format

- Fibertree in stream format

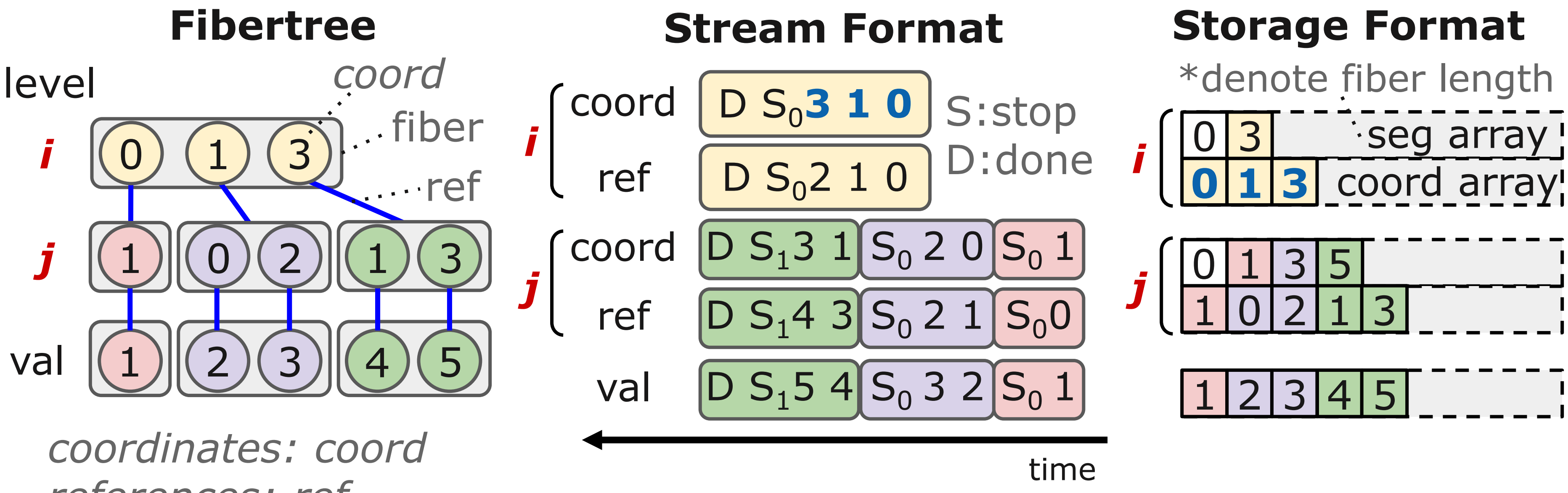


coordinates: coord
references: ref
segments: seg
values: val



Stream and Storage Format

- Fibertree in storage format

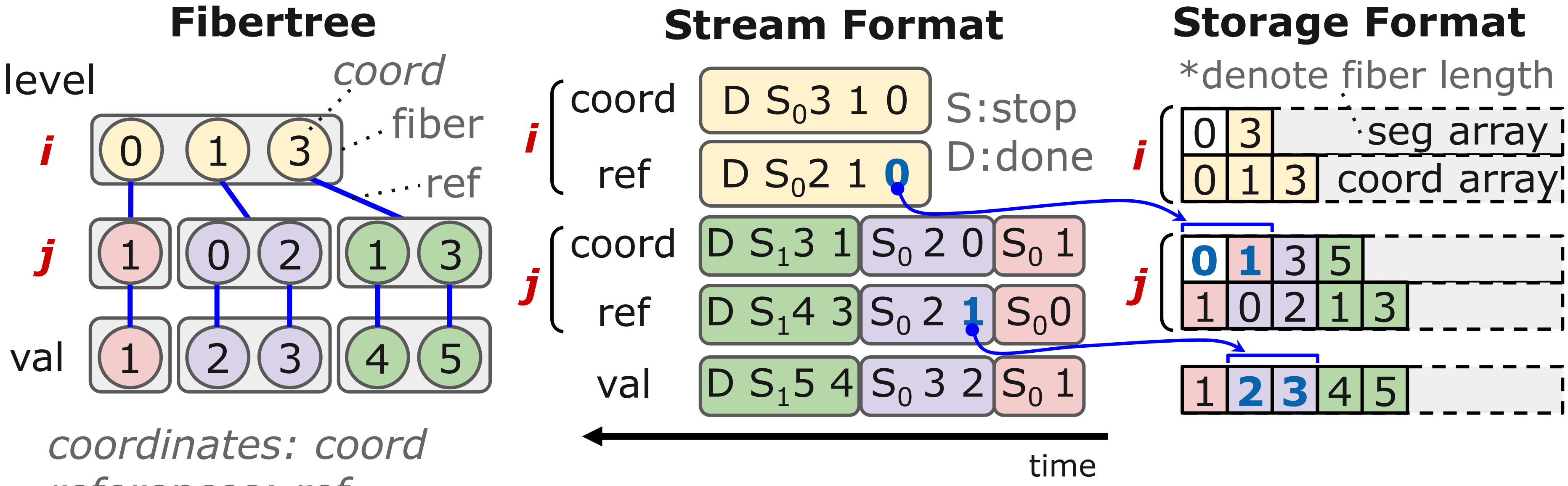


coordinates: coord
references: ref
segments: seg

values: val

Stream and Storage Format

- Fibertree in storage format



coordinates: coord

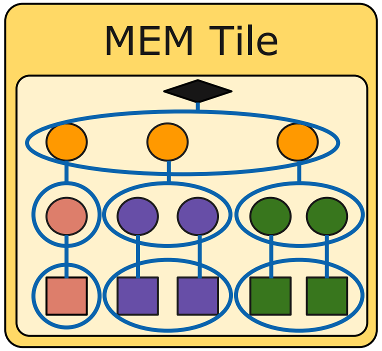
references: ref

segments: seg

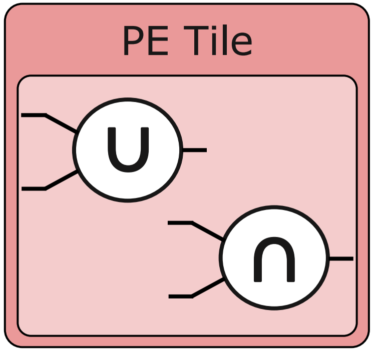
values: val

Sparse Acceleration Hardware

- MEMs: sparse memory controller logic to convert between the stream and storage representation of fibertrees
- PEs: contains logic to eliminate all ineffectual compute and support tensor algebra



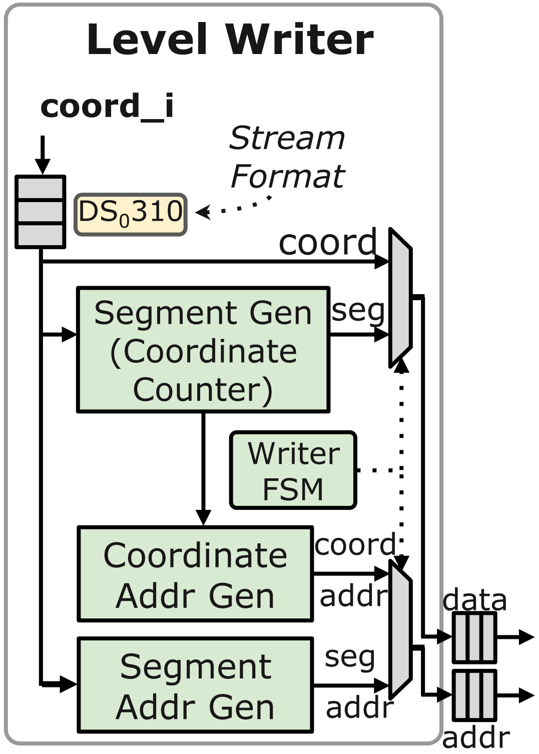
Level Writer
Level Scanner
Level Buffer

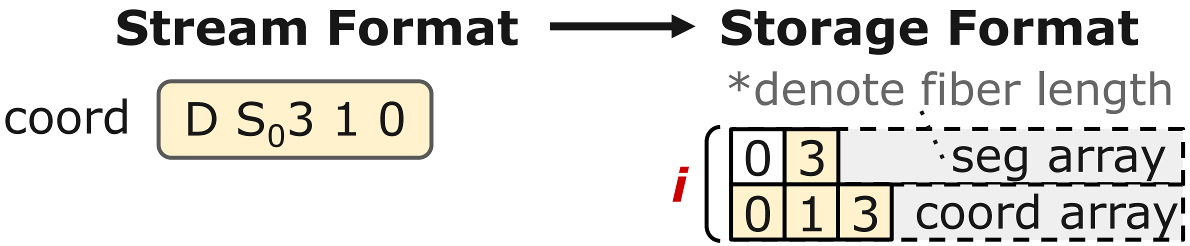


Intersecter
Unioner
Coordinate Dropper
Reducer
Repeater

Primitives in MEM Tile - Level Writer

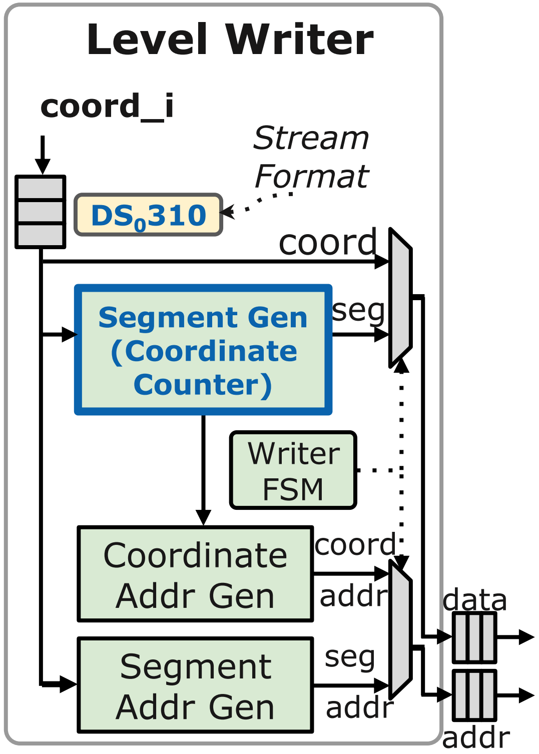
- The level writer converts a stream to the storage format

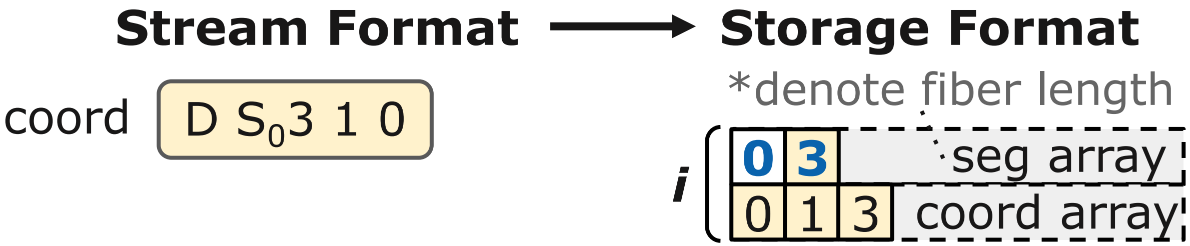




Primitives in MEM Tile - Level Writer

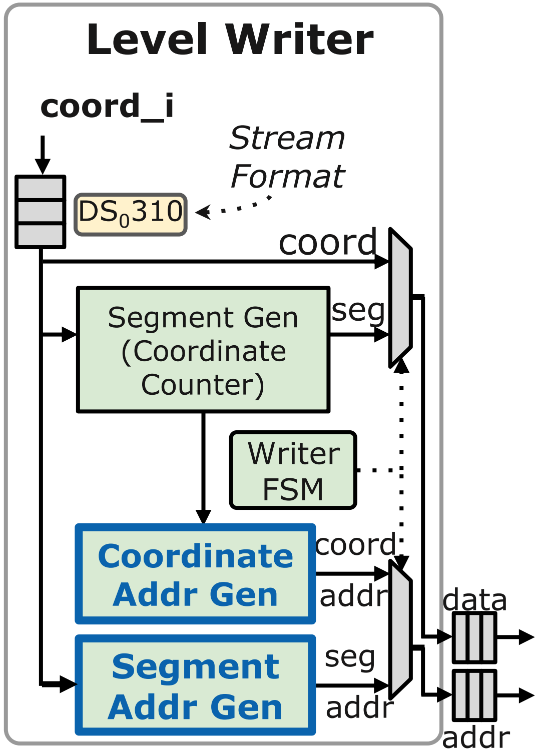
- The level writer converts a stream to the storage format

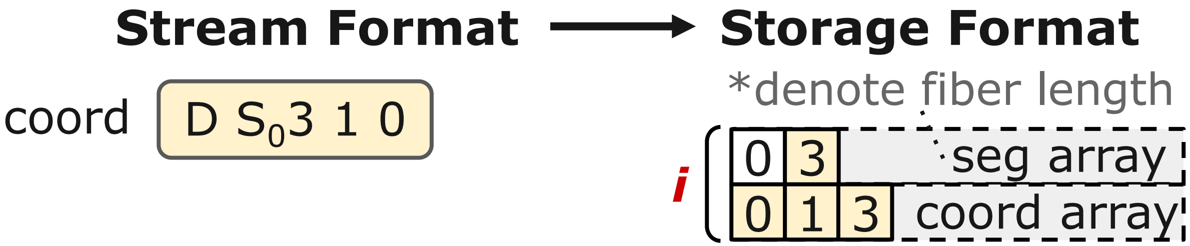




Primitives in MEM Tile - Level Writer

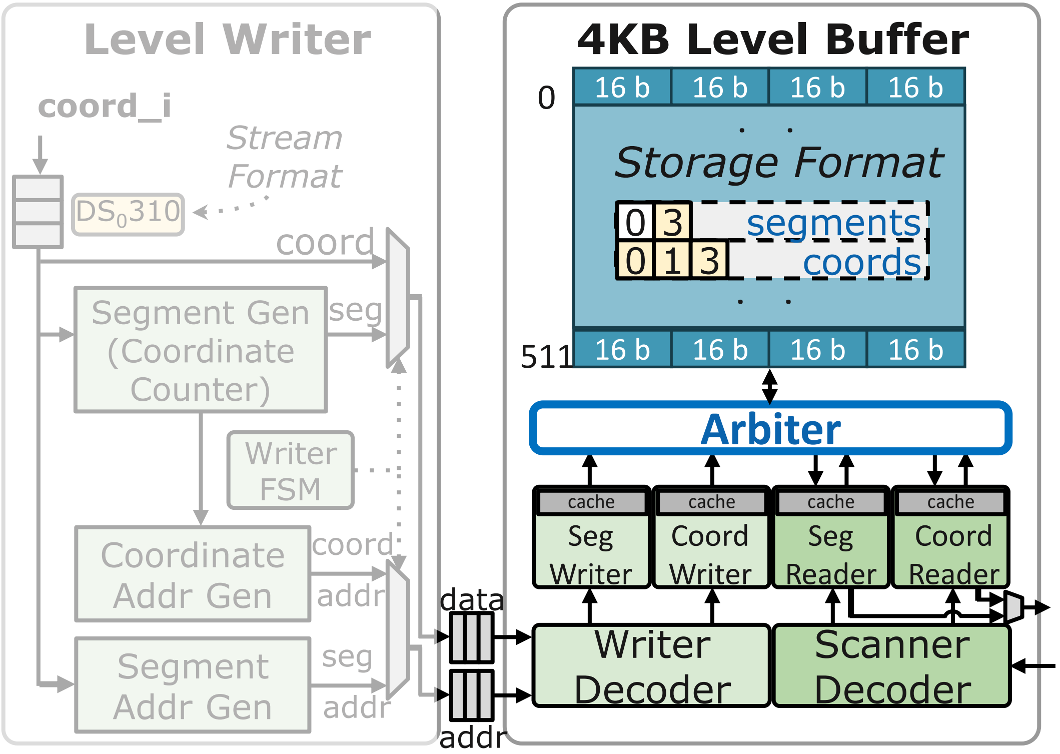
- The level writer converts a stream to the storage format





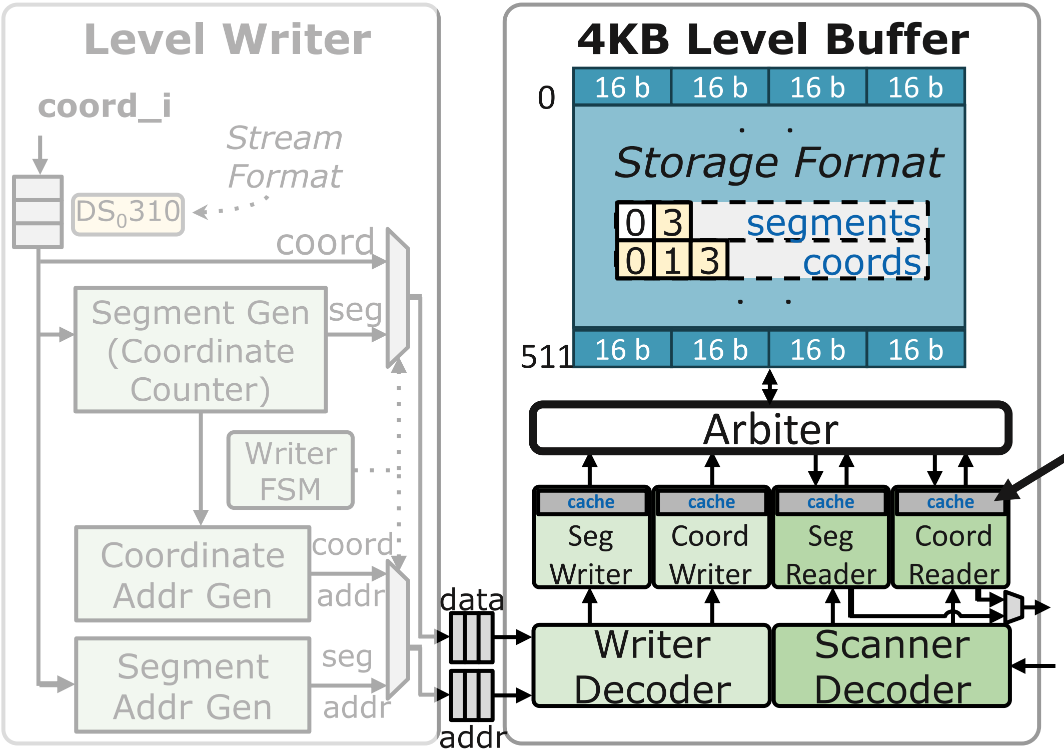
Primitives in MEM Tile - Level Buffer

- The level buffer arbitrates writes/reads with support for caching



Primitives in MEM Tile – Level Buffer

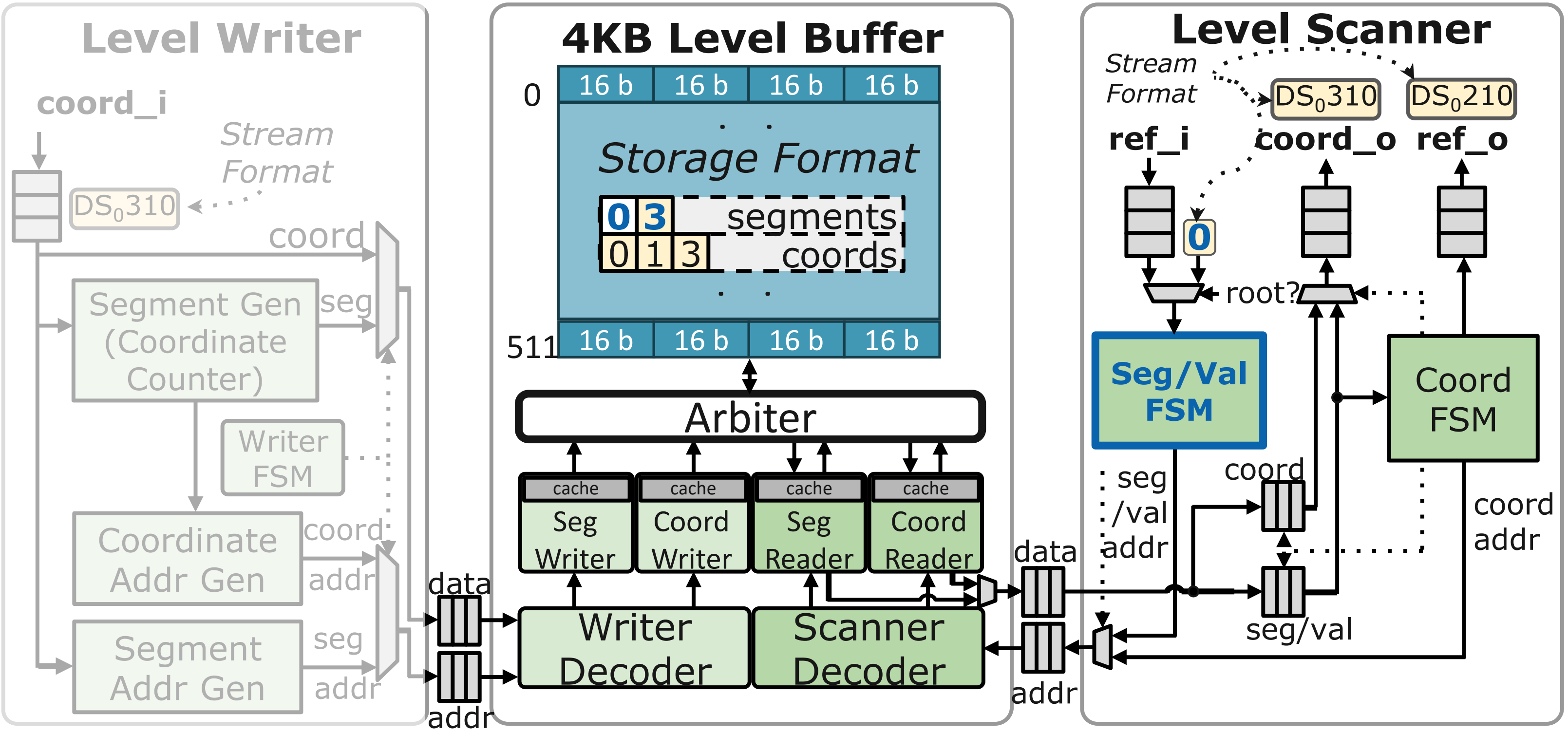
- The level buffer arbitrates writes/reads with support for caching



Avoid redundant reads of the same 4-element word

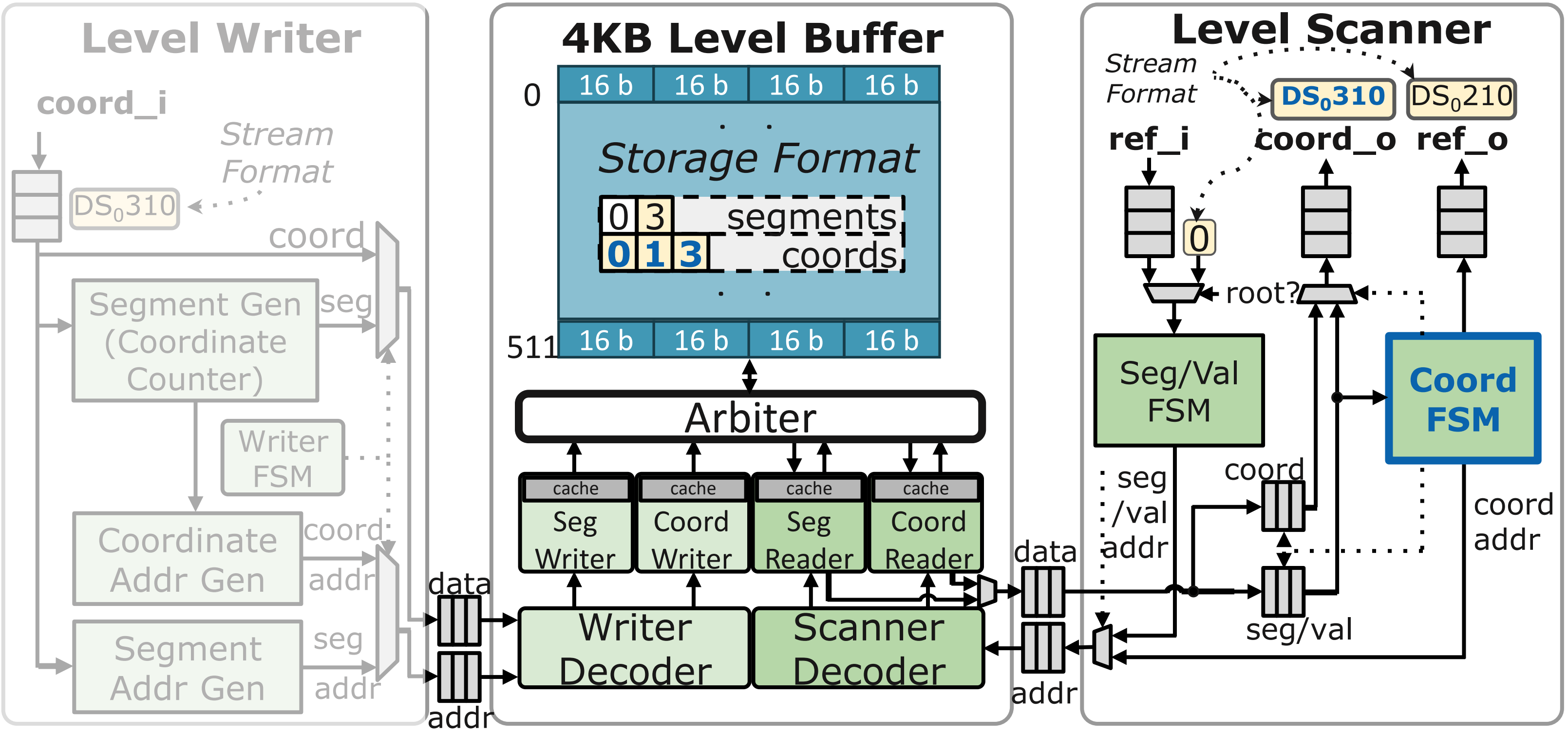
Primitives in MEM Tile - Level Scanner

- The level scanner produces the next level fiber for a reference



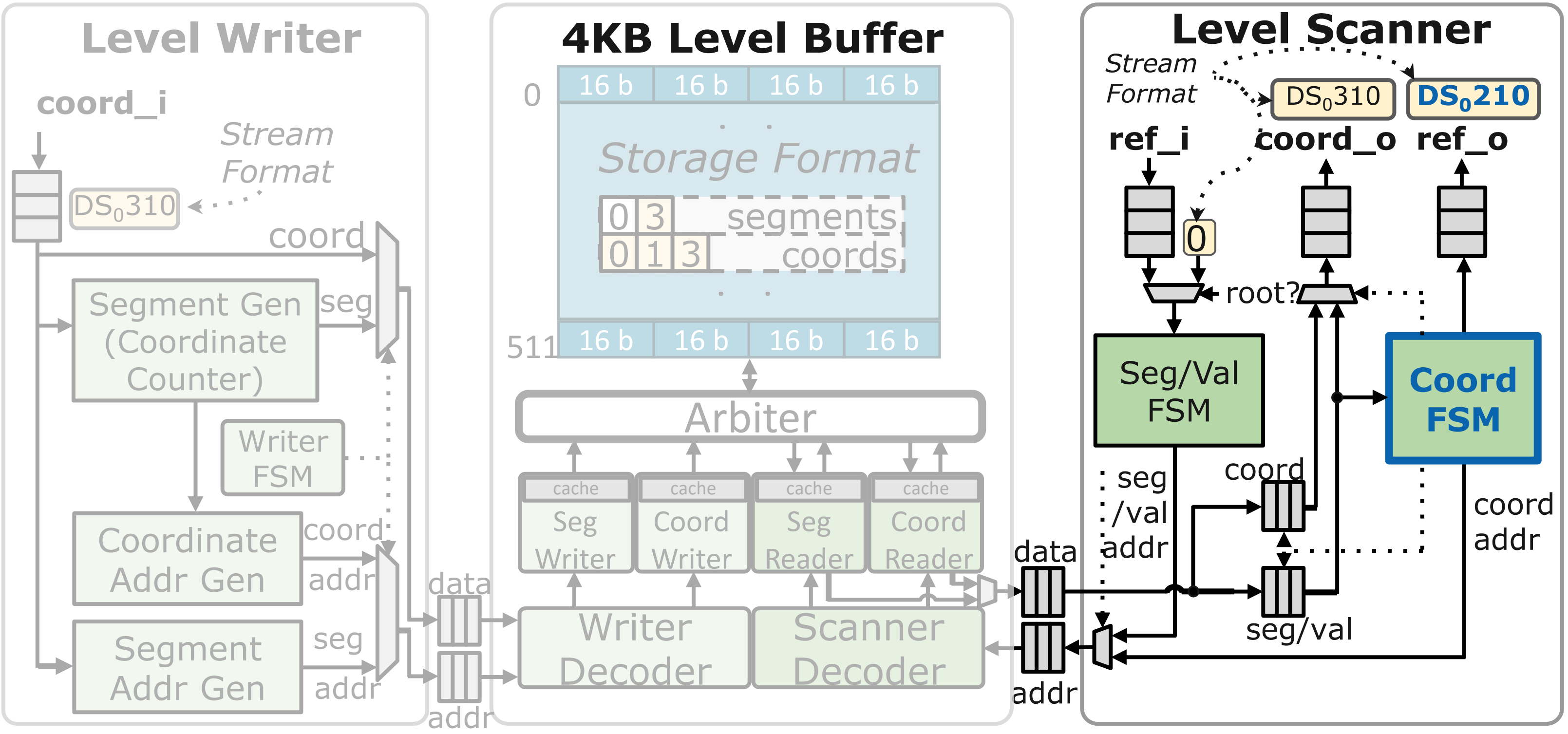
Primitives in MEM Tile - Level Scanner

- The level scanner produces the next level fiber for a reference



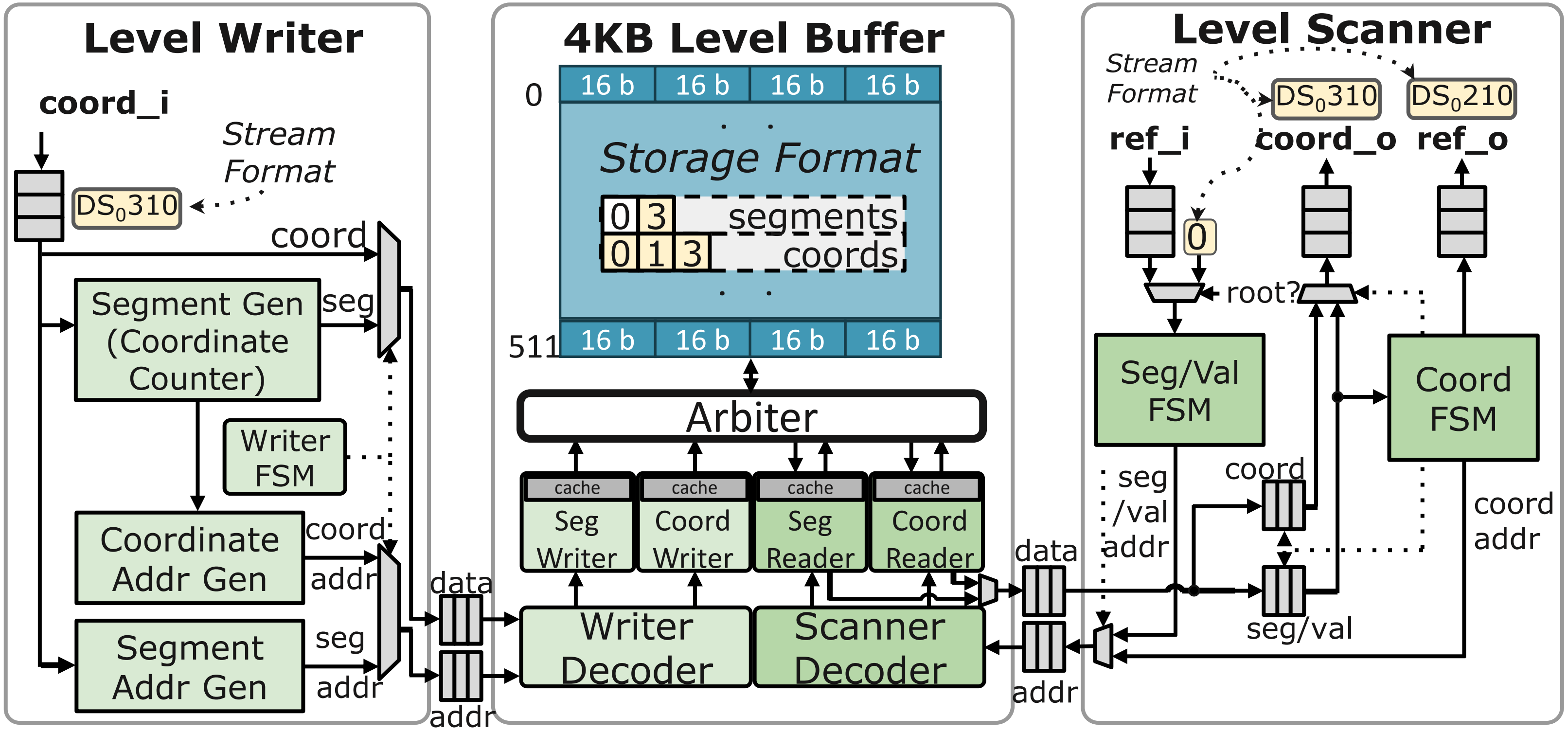
Primitives in MEM Tile - Level Scanner

- The level scanner produces the next level fiber for a reference



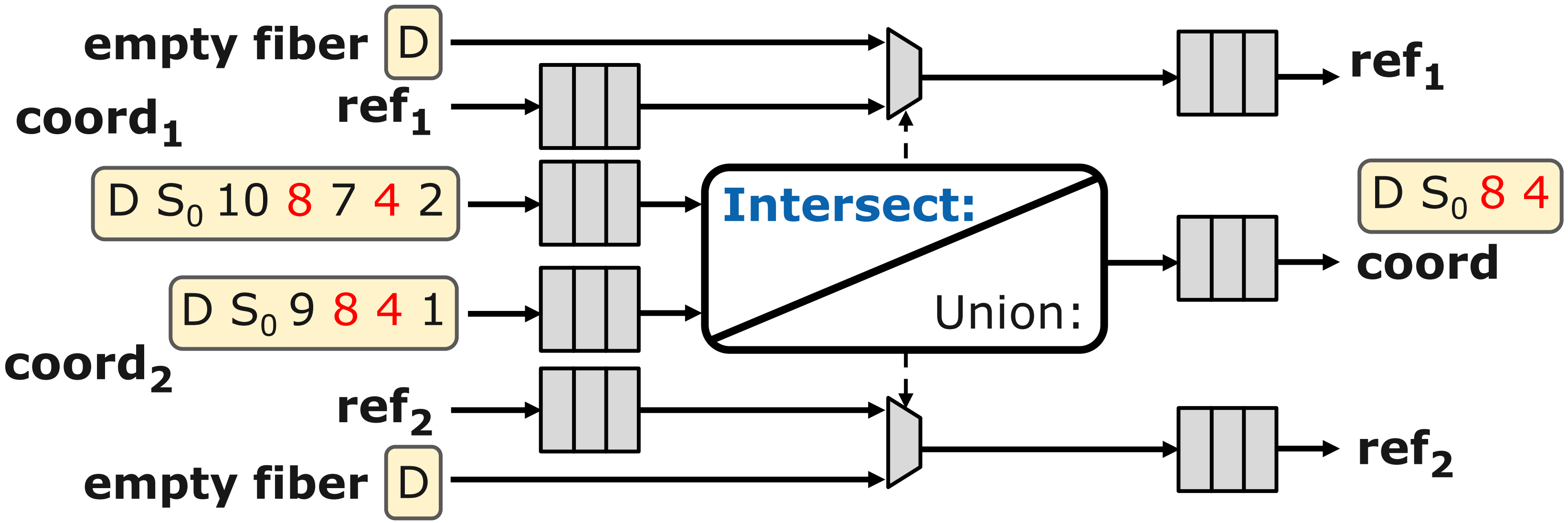
Primitives in MEM Tile

- Store and process any-dimensional tensors



Primitives in PE Tile - Intersecter / Unioner

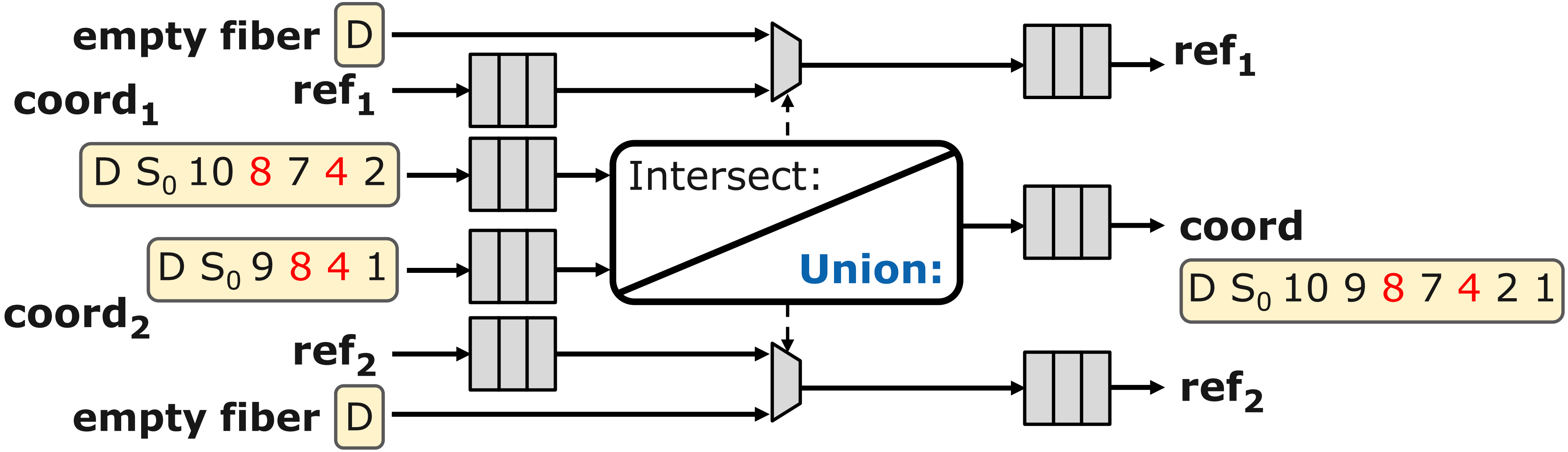
- The intersecter compares and keeps equivalent coordinates



Stanford University

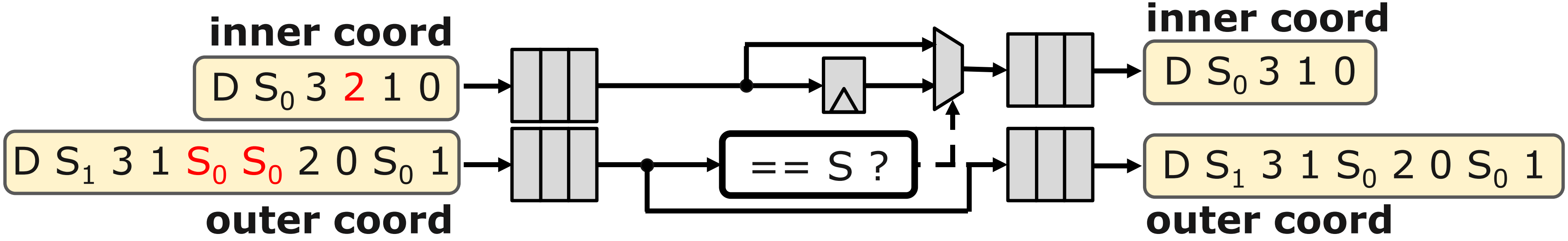
Primitives in PE Tile - Intersecter / Unioner

- The unioner collects all incoming coordinates



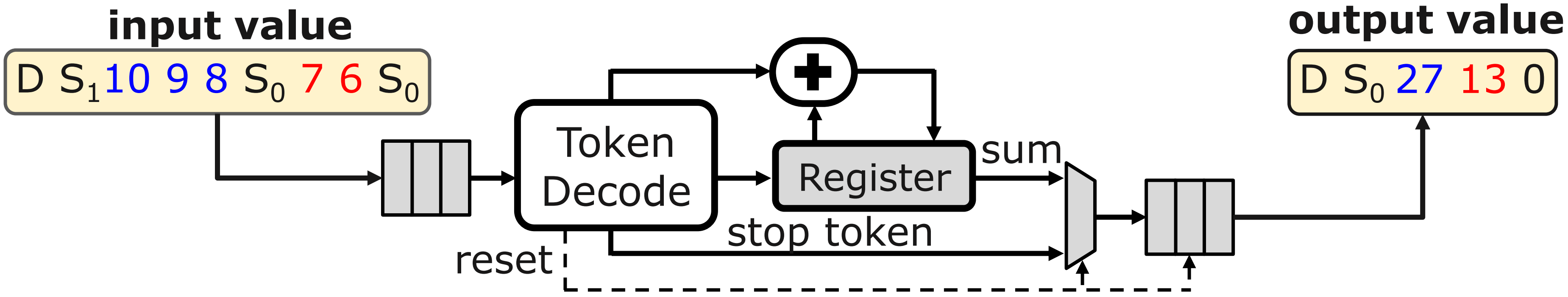
Primitives in PE Tile - Coordinate Dropper

- The coordinate dropper removes empty fibers after an intersect



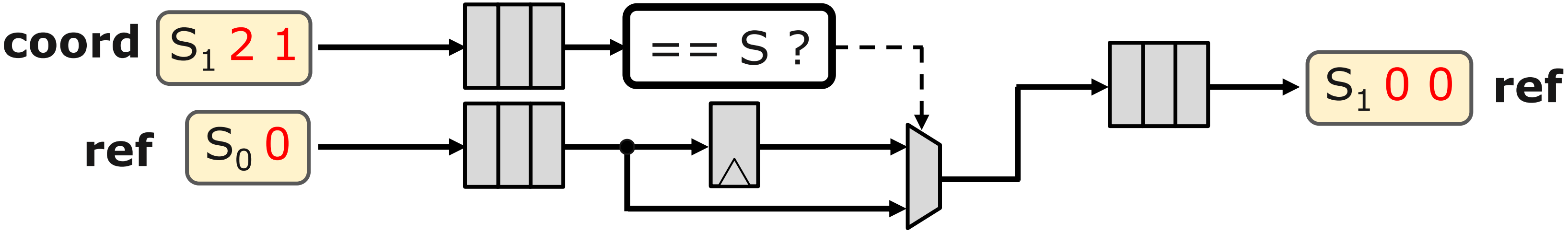
Primitives in PE Tile - Reducer

- The reducer accumulates over a fiber

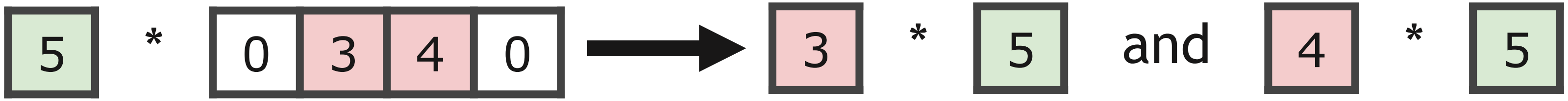


Primitives in PE Tile - Repeater

- The repeater broadcasts a stream over another



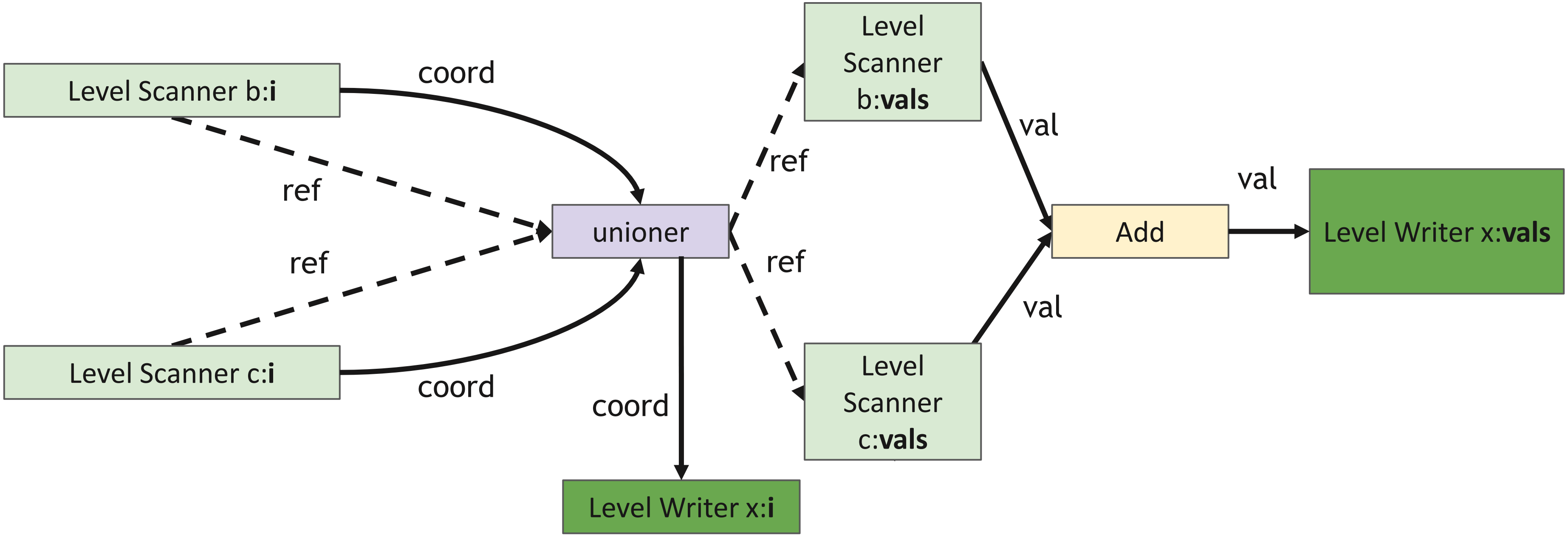
Ex: Scalar Broadcasted over a Vector



Composable Primitives

- With these primitives we support all of sparse tensor algebra

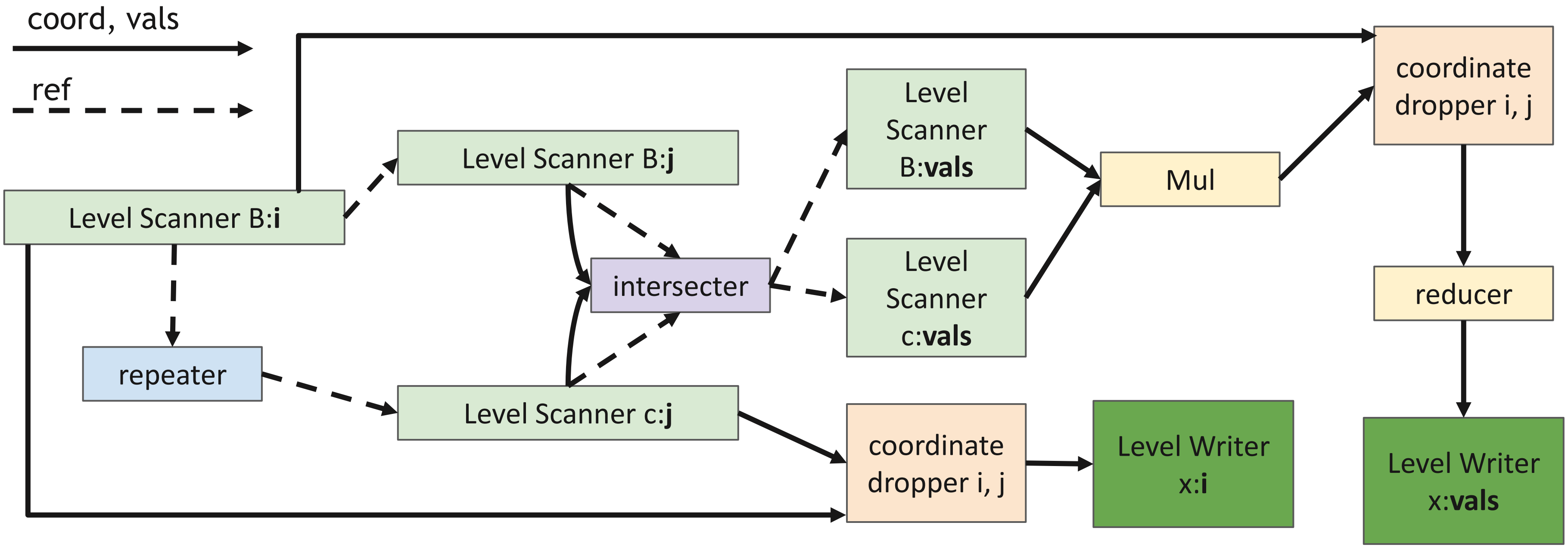
Vector-Vector Element Add ($X_i = b_i + c_i$)



Composable Primitives

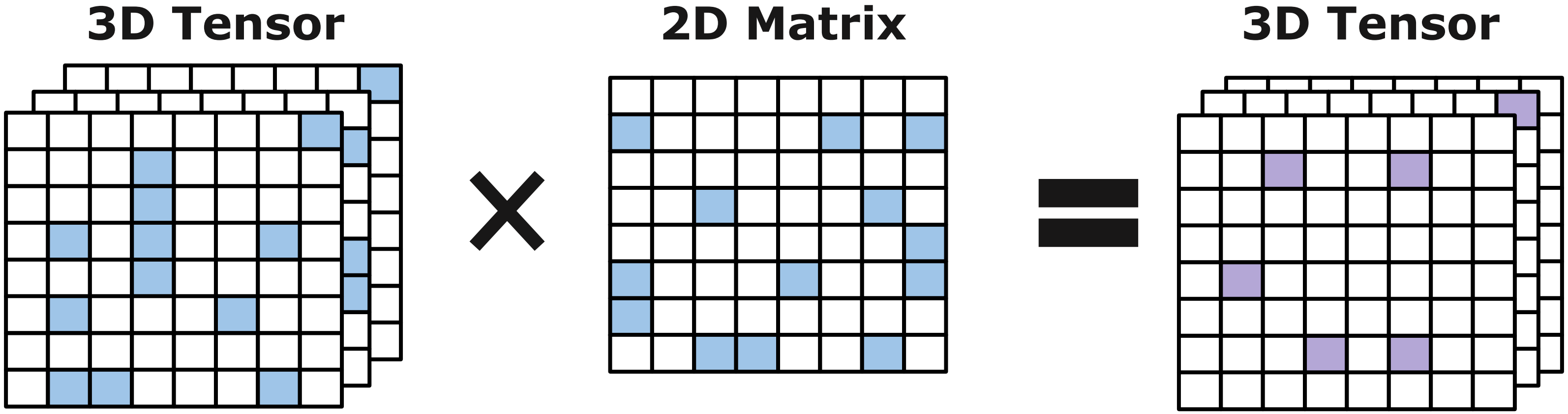
- With these primitives we support all of sparse tensor algebra

Ex: Matrix-Vector Multiplication ($X_i = \sum_j B_{ij} c_j$)



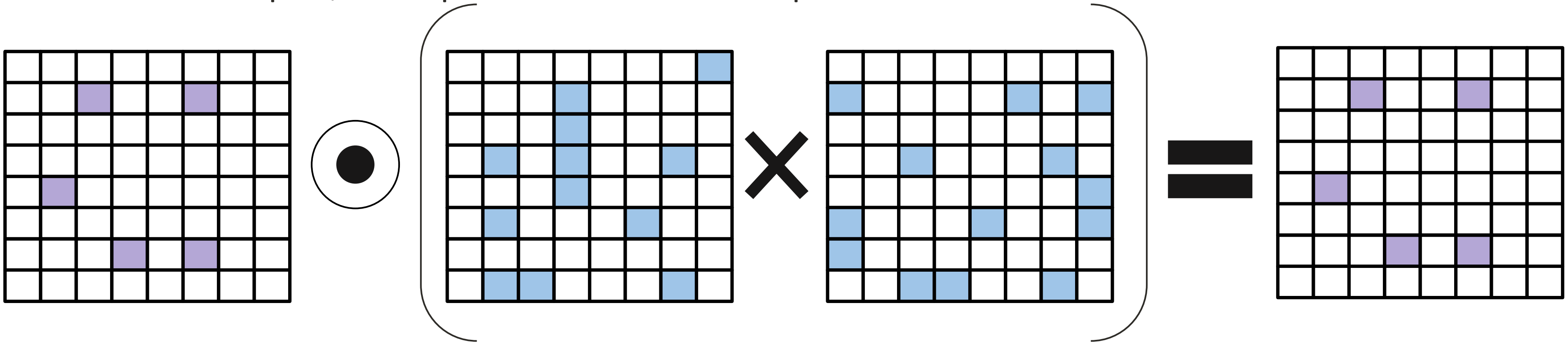
Composable Primitives

- Utilizing these composable primitives, we support **high-dimensional tensors** and multiple inputs with fusion
- For example, tensor times matrix:



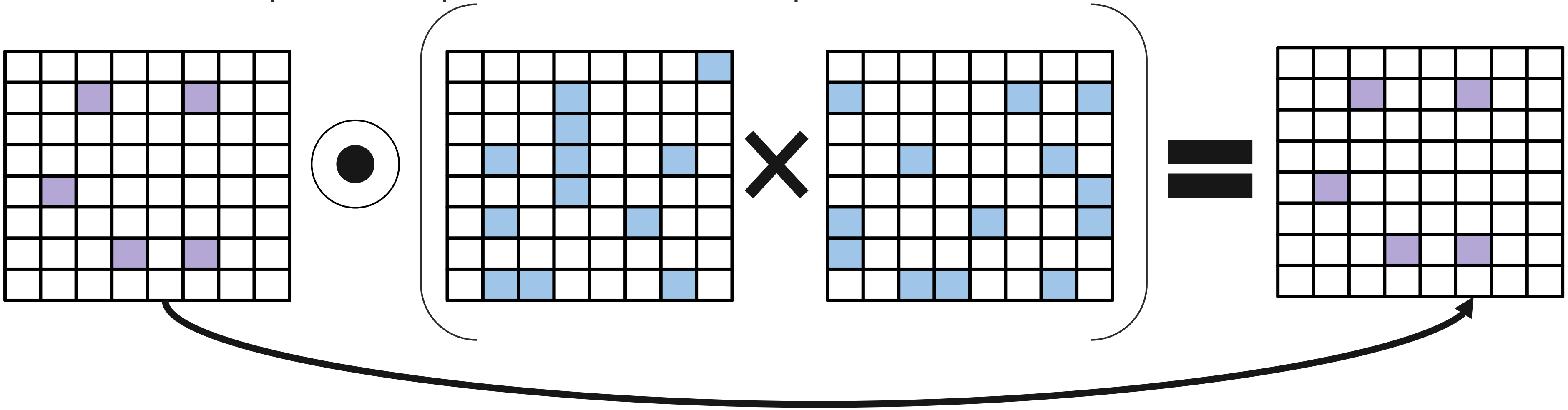
Composable Primitives

- Utilizing these composable primitives, we support high-dimensional tensors and **multiple inputs with fusion**
- For example, sampled matrix multiplication:



Composable Primitives

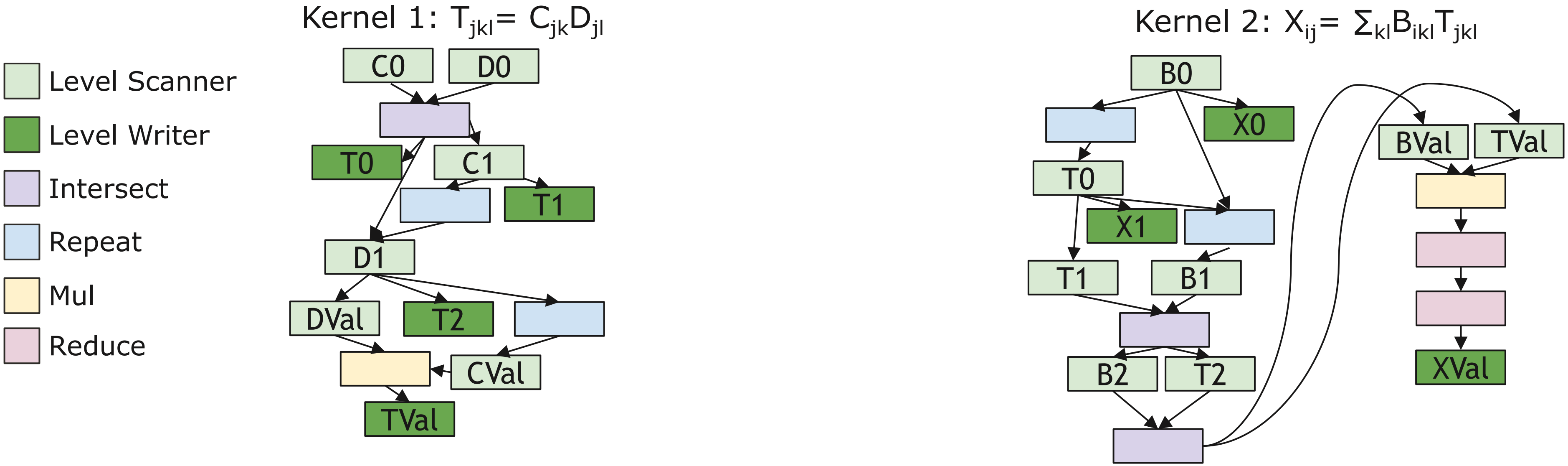
- Utilizing these composable primitives, we support high-dimensional tensors and **multiple inputs with fusion**
- For example, sampled matrix multiplication:



Only calculate necessary indices!

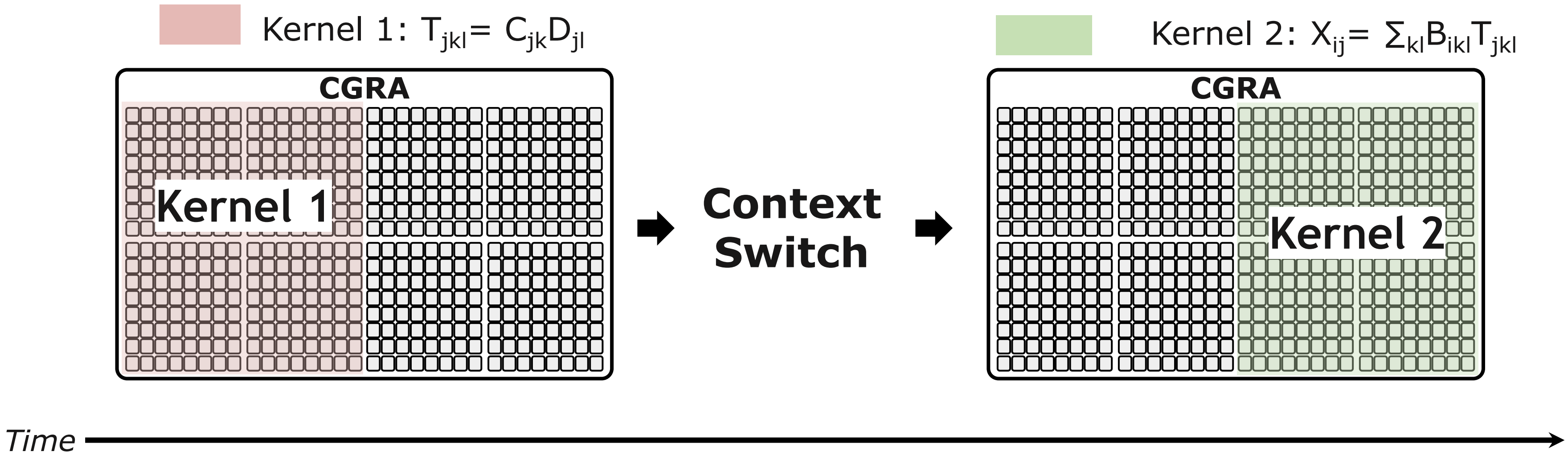
Without Fusion

- For example, for a higher-dimensional, multi-input expression like **Matricized Tensor Times Khatri-Rao Product** (MTTKRP: $X_{ij}=\sum_{kl}B_{ikl}C_{jk}D_{jl}$)
- An accelerator with 2-input support splits MTTKRP into Kernel 1: $T_{jkl}= C_{jk}D_{jl}$ and Kernel 2: $X_{ij}= \sum_{kl}B_{ikl}T_{jkl}$



Without Fusion

- For example, for a higher-dimensional, multi-input expression like **Matricized Tensor Times Khatri-Rao Product** (MTTKRP: $X_{ij}=\sum_{kl}B_{ikl}C_{jk}D_{jl}$)
- An accelerator with 2-input support splits MTTKRP into Kernel 1: $T_{jkl}= C_{jk}D_{jl}$ and Kernel 2: $X_{ij}= \sum_{kl}B_{ikl}T_{jkl}$

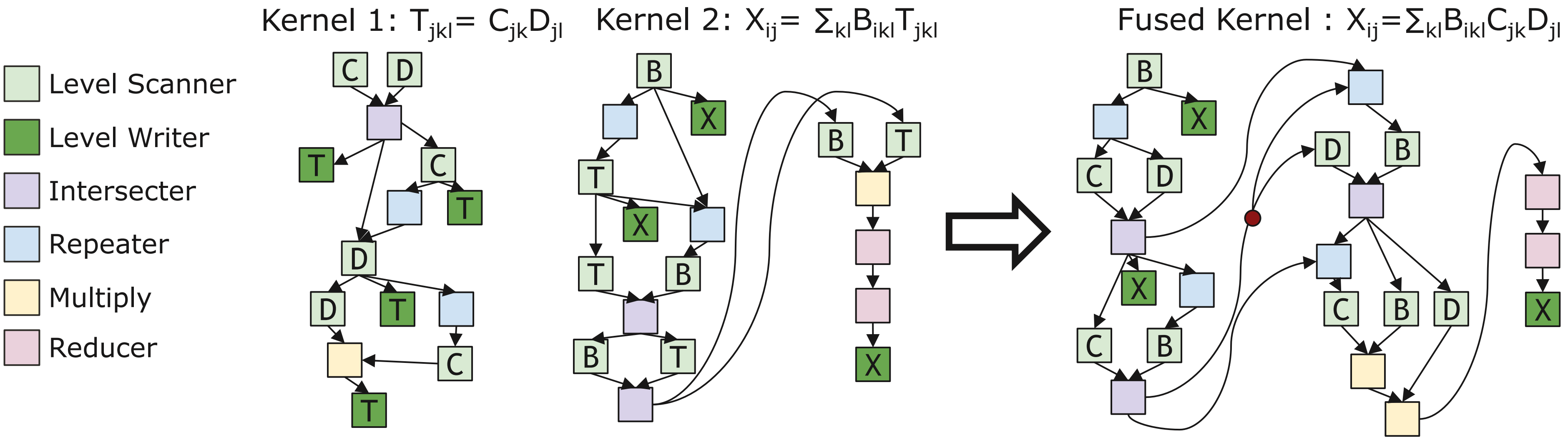


🟥 Kernel 1: $T_{jkl}= C_{jk}D_{jl}$

🟩 Kernel 2: $X_{ij}= \sum_{kl}B_{ikl}T_{jkl}$

Fusion Results

- With Onyx we use a single **fused** kernel on the array



Kernel 1: $T_{jkl} = C_{jk}D_{jl}$

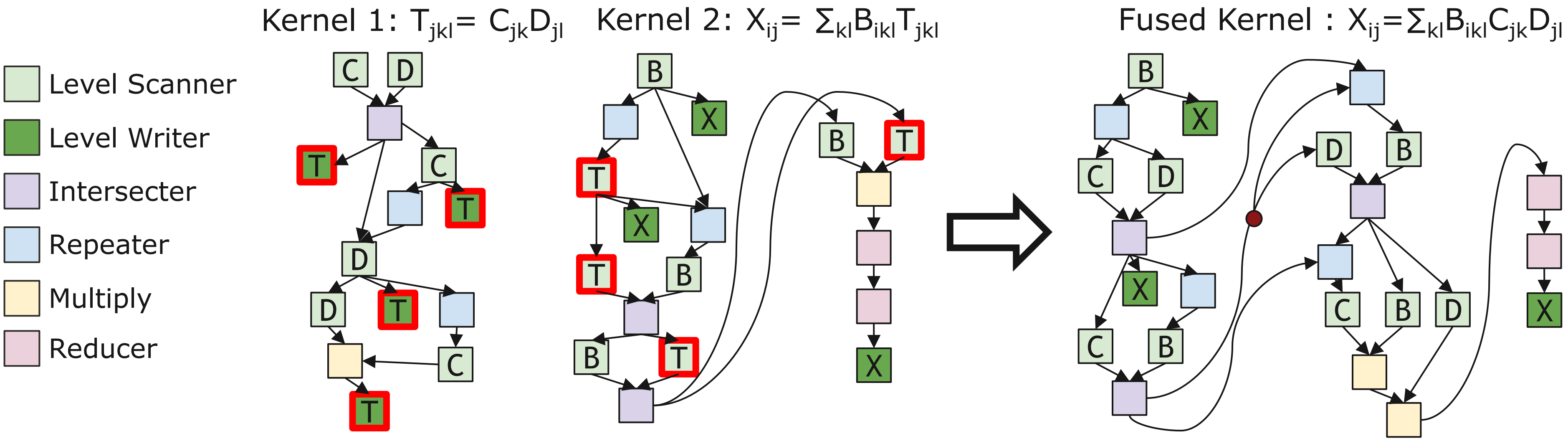
Kernel 2: $X_{ij} = \Sigma_{kl}B_{ikl}T_{jkl}$

Fused Kernel : $X_{ij} = \Sigma_{kl}B_{ikl}C_{jk}D_{jl}$

- 🟩 Level Scanner
- 🟩 Level Writer
- 🟪 Intersecter
- 🟦 Repeater
- 🟨 Multiply
- 🟥 Reducer

Fusion Results

- With Onyx we use a single <u>**fused**</u> kernel on the array
- Eliminate intermediate storage of T



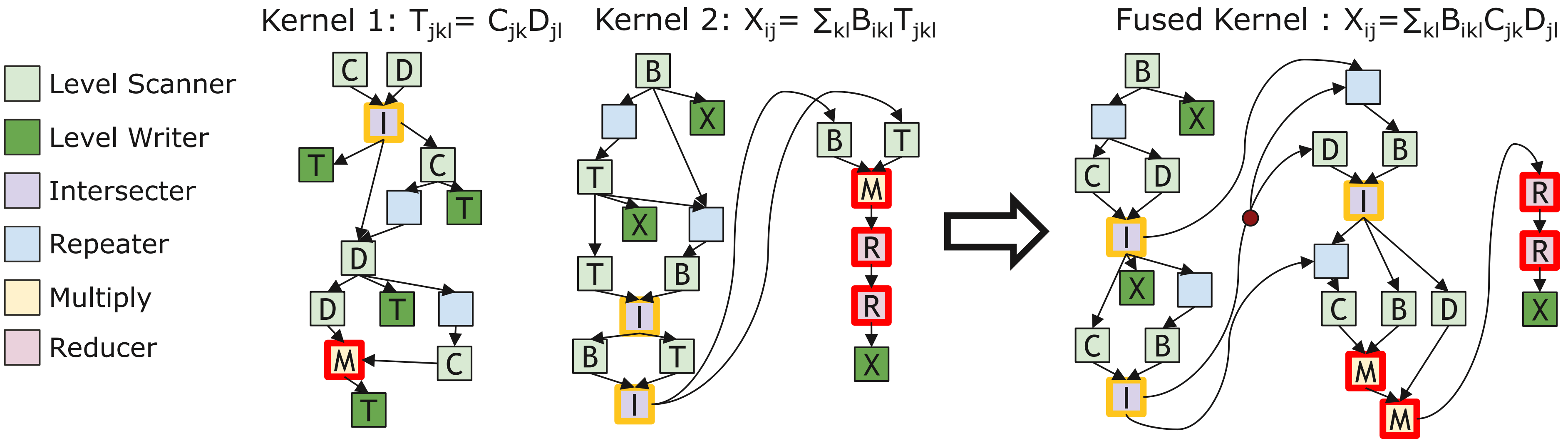
Kernel 1: $T_{jkl} = C_{jk}D_{jl}$

Kernel 2: $X_{ij} = \Sigma_{kl}B_{ikl}T_{jkl}$

Fused Kernel : $X_{ij} = \Sigma_{kl}B_{ikl}C_{jk}D_{jl}$

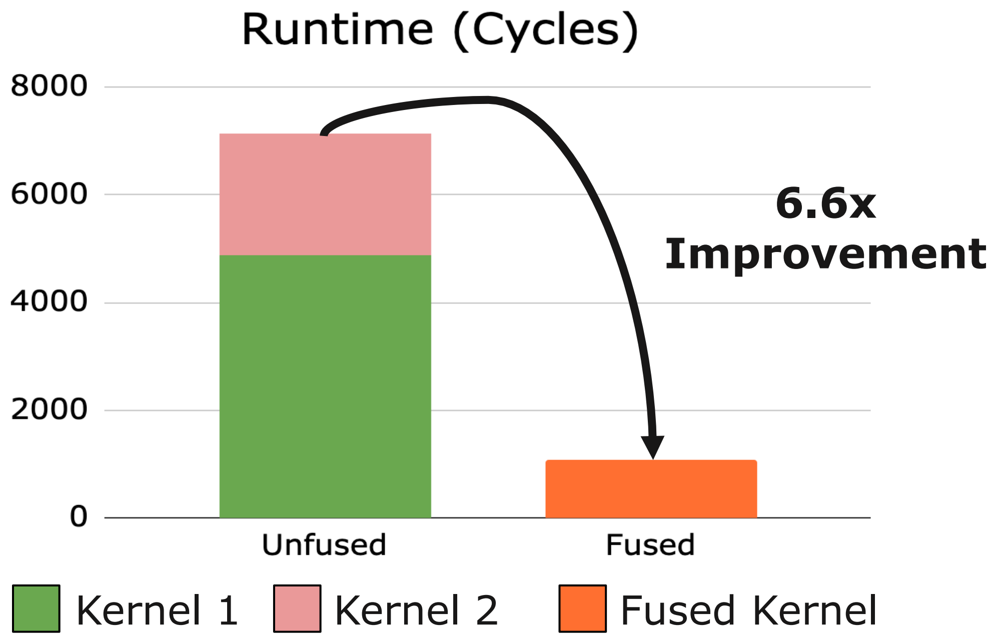
Fusion Results

- With Onyx we use a single <u>fused</u> kernel on the array
- Perform all intersections before compute


Kernel 1: $T_{jkl} = C_{jk}D_{jl}$
Kernel 2: $X_{ij} = \Sigma_{kl}B_{ikl}T_{jkl}$
Fused Kernel : $X_{ij} = \Sigma_{kl}B_{ikl}C_{jk}D_{jl}$
Level Scanner
Level Writer
Intersecter
Repeater
Multiply
Reducer

MTTKRP Fusion Results

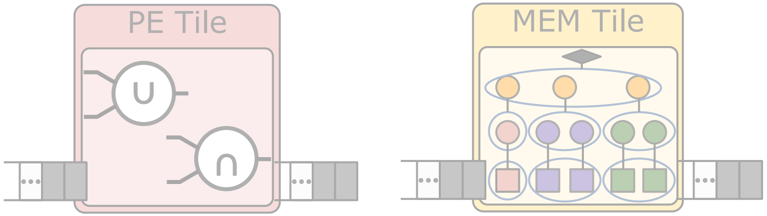
- Onyx supports a single **fused** kernel on the array
- Avoids intermediate storage and eliminates unnecessary compute



Contributions

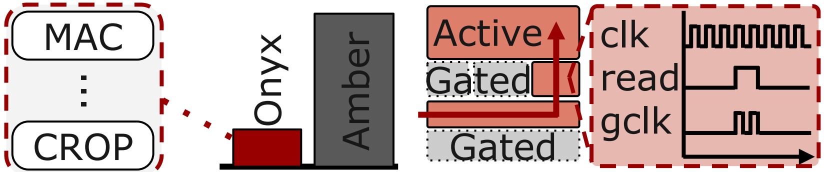
Sparse Acceleration Hardware

Composable primitives accelerating arbitrary sparse tensor algebra kernels



Dense Acceleration Improvements

Compute and memory controller optimizations for dense applications



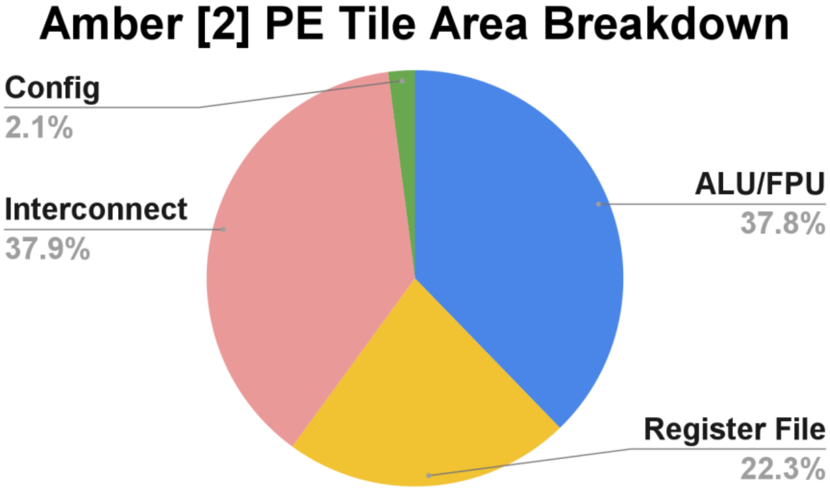
New PE OPs

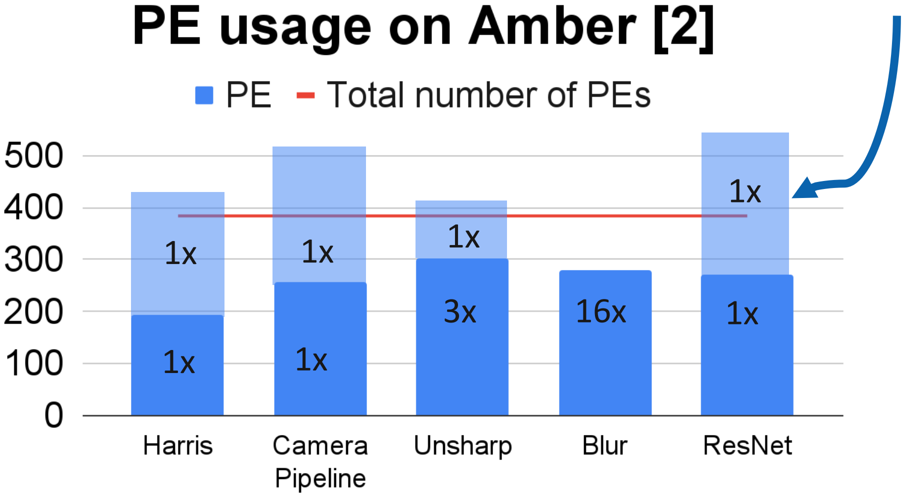
PE Usage per App

Dynamic-/Static- GLB Clock Gating

Challenge 1: Compute Density

- Compute only 37.8% of PE area
- Since each app requires several simple PEs, it is difficult to further unroll applications - low CGRA PE utilization

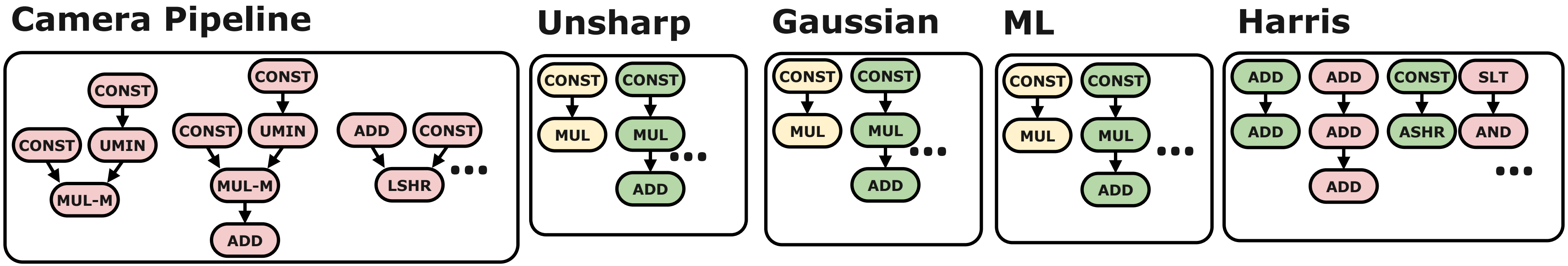




[2] K. Feng et al., JSSC'2023.

Optimized Compute

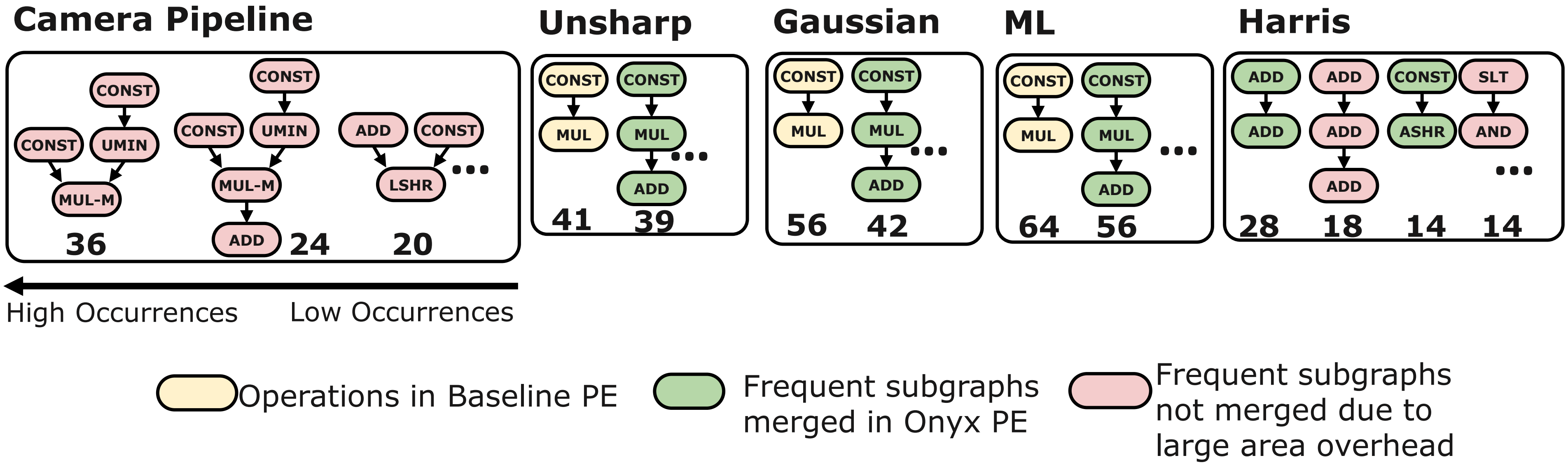
- Utilized the Automated PE Exploration tool (APEX [3])
- Step 1: Mine frequent subgraphs from each application



[3] J. Melchert et al., ASPLOS'2023.

Optimized Compute

- Step 1: Mine frequent subgraphs from each application
- Step 2: Order subgraphs by # non-overlapping occurrences using maximal independent set analysis



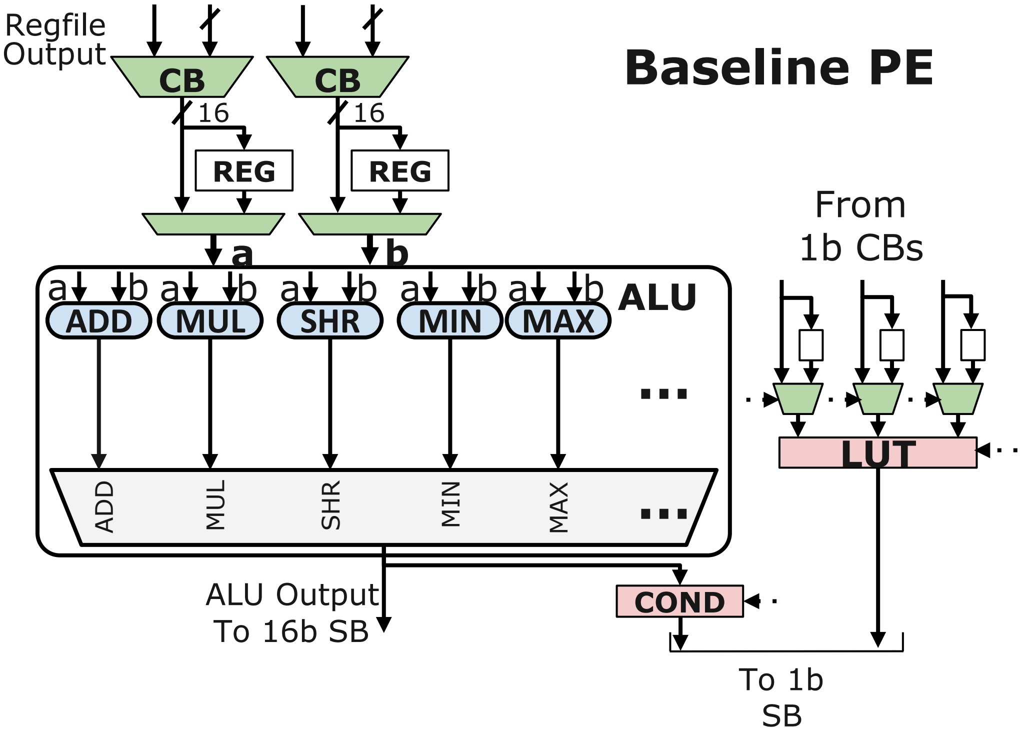
🟨 Operations in Baseline PE

🟩 Frequent subgraphs merged in Onyx PE

🟥 Frequent subgraphs not merged due to large area overhead

Optimized Compute

- Step 3: Merge subgraphs into PE



Baseline PE

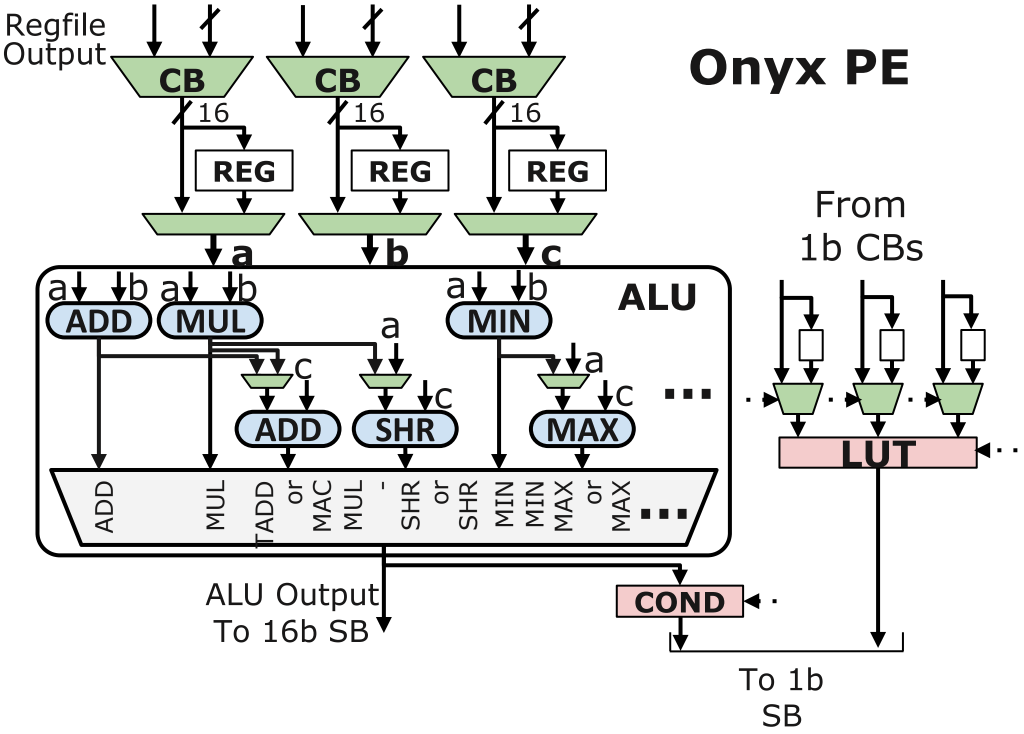
PE Instruction Set:

INT16/BIT: ADC SBC ABS MUX MULT0 MULT1 MULT2 SHR SHL OR AND NOT XOR MIN MAX EQ CMP

BFloat16: ADD SUB CMP MUL GETMAN* ADDIEXP* SUBEXP* EXP2F* F2INT* GETFR* INT2F* (*Used in complex ops)

Optimized Compute

- Step 3: Merge subgraphs into PE



PE Instruction Set:

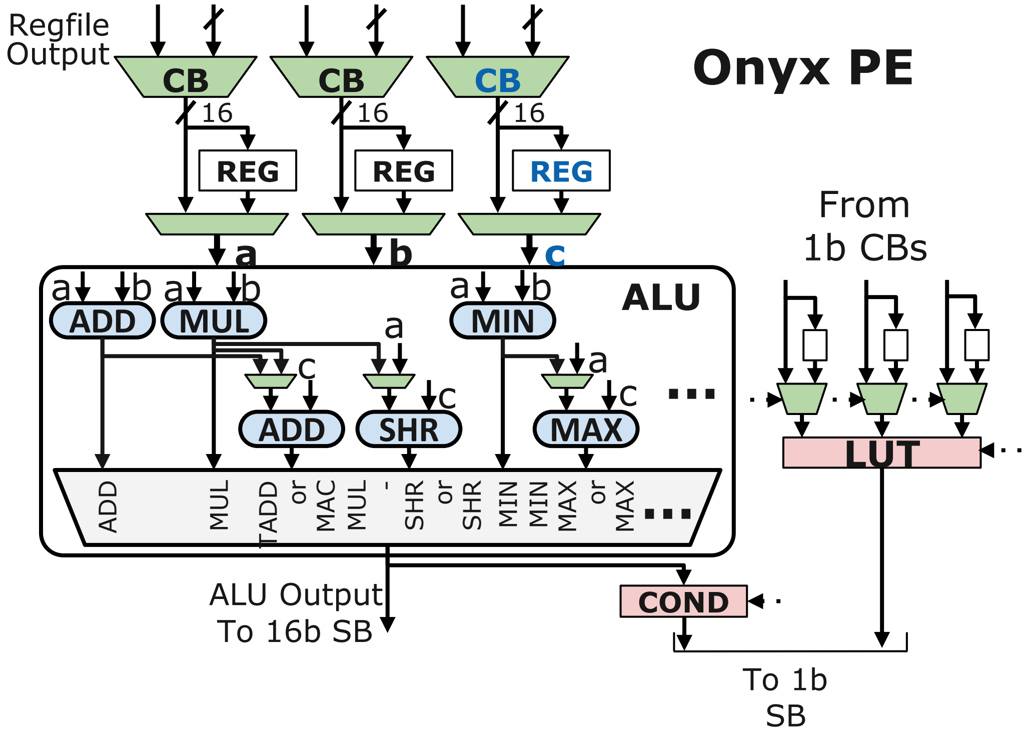
INT16/BIT: ADC SBC ABS MUX MULT0 MULT1 MULT2 SHR SHL OR AND NOT XOR MIN MAX EQ CMP

MAC variants (MULADD MULSUB) TADD variants (ADDADD ADDSUB SUBADD SUBSUB) MINMAX MULSHR

BFloat16: ADD SUB CMP MUL GETMAN* ADDIEXP* SUBEXP* EXP2F* F2INT* GETFR* INT2F* (*Used in complex ops)

Optimized Compute

- Step 3: Merge subgraphs into PE



PE Instruction Set:

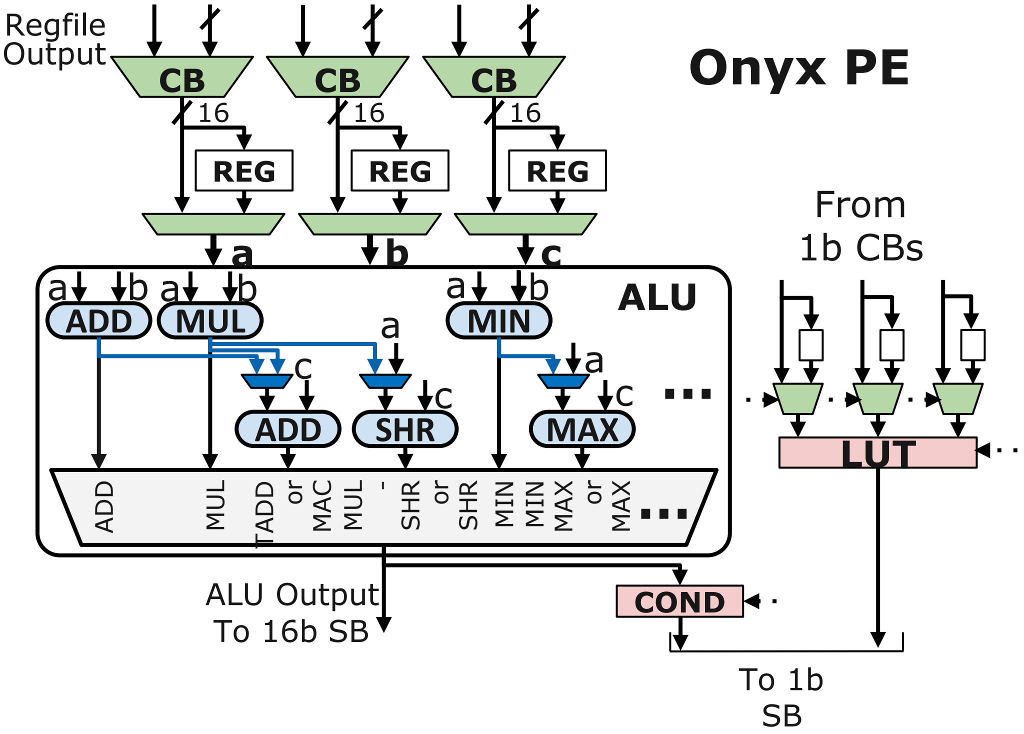
INT16/BIT: ADC SBC ABS MUX MULT0 MULT1 MULT2 SHR SHL OR AND NOT XOR MIN MAX EQ CMP

MAC variants (MULADD MULSUB) TADD variants (ADDADD ADDSUB SUBADD SUBSUB) MINMAX MULSHR

BFloat16: ADD SUB CMP MUL GETMAN* ADDIEXP* SUBEXP* EXP2F* F2INT* GETFR* INT2F* (*Used in complex ops)

Optimized Compute

- Step 3: Merge subgraphs into PE



PE Instruction Set:

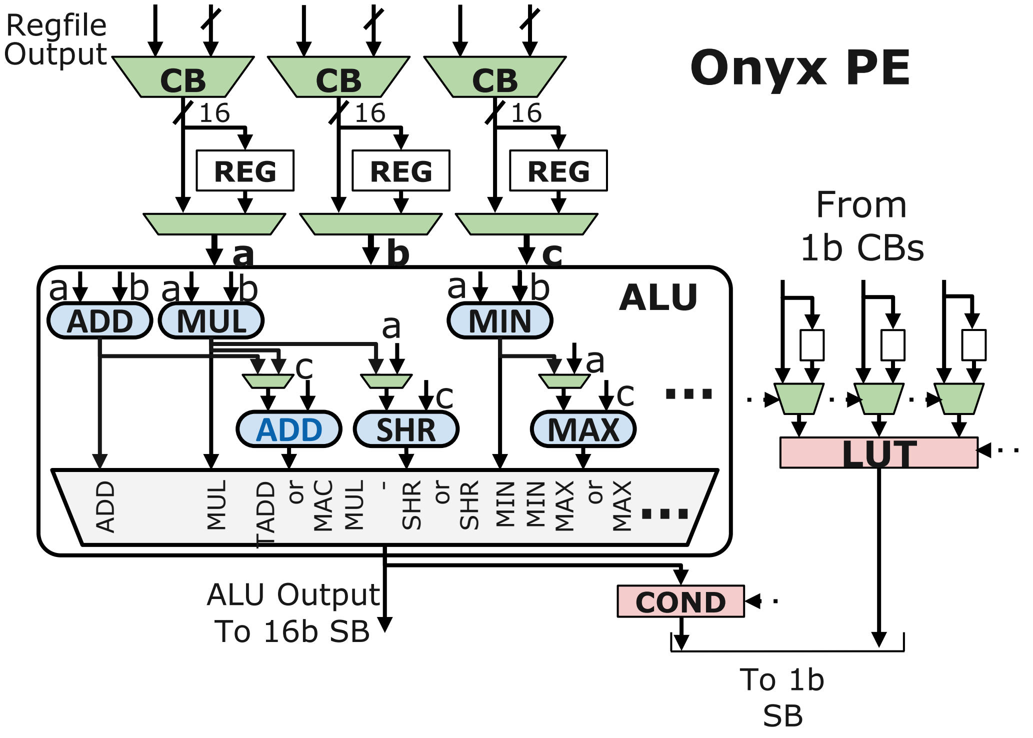
INT16/BIT: ADC SBC ABS MUX MULT0 MULT1 MULT2 SHR SHL OR AND NOT XOR MIN MAX EQ CMP

MAC variants (MULADD MULSUB) TADD variants (ADDADD ADDSUB SUBADD SUBSUB) MINMAX MULSHR

BFloat16: ADD SUB CMP MUL GETMAN* ADDIEXP* SUBEXP* EXP2F* F2INT* GETFR* INT2F* (*Used in complex ops)

Optimized Compute

- Step 3: Merge subgraphs into PE



PE Instruction Set:

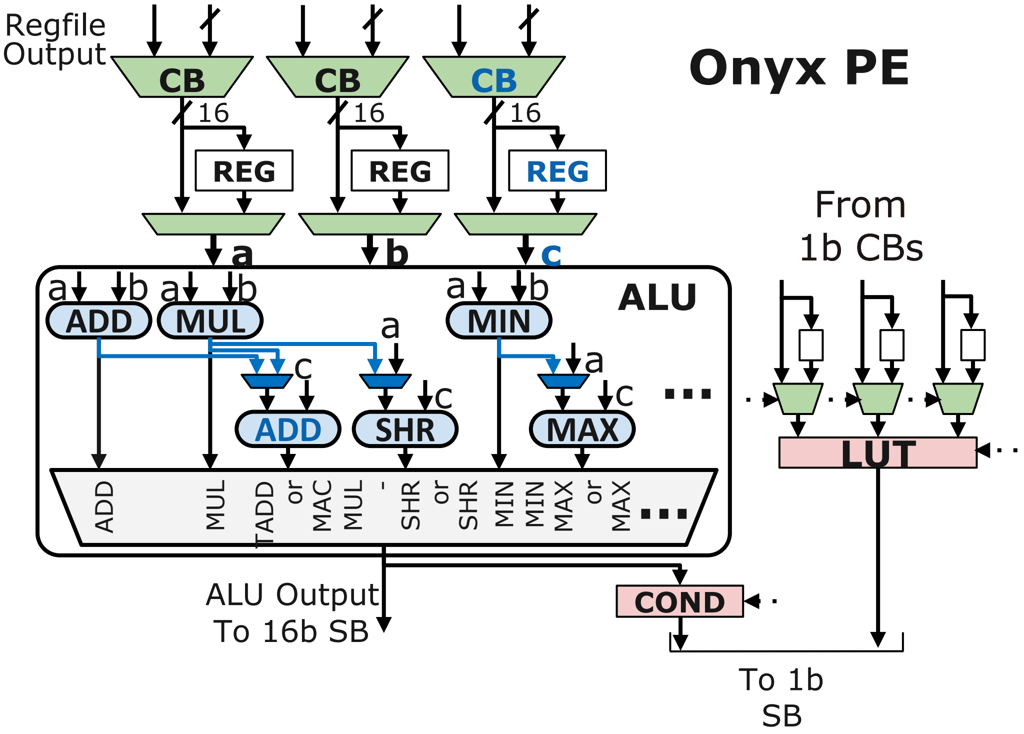
INT16/BIT: ADC SBC ABS MUX MULT0 MULT1 MULT2 SHR SHL OR AND NOT XOR MIN MAX EQ CMP

MAC variants (MULADD MULSUB) TADD variants (ADDADD ADDSUB SUBADD SUBSUB) MINMAX MULSHR

BFloat16: ADD SUB CMP MUL GETMAN* ADDIEXP* SUBEXP* EXP2F* F2INT* GETFR* INT2F* (*Used in complex ops)

Optimized Compute

- Step 3: Merge subgraphs into PE



PE Instruction Set:

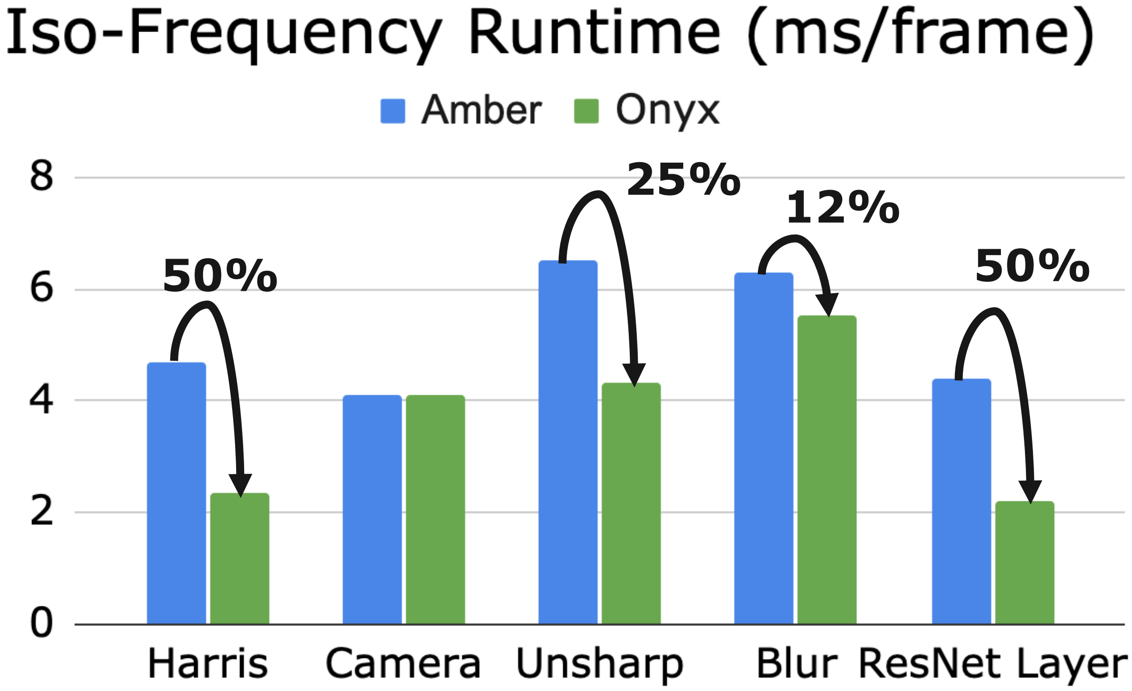
INT16/BIT: ADC SBC ABS MUX MULT0 MULT1 MULT2 SHR SHL OR AND NOT XOR MIN MAX EQ CMP

MAC variants (MULADD MULSUB) TADD variants (ADDADD ADDSUB SUBADD SUBSUB) MINMAX MULSHR

BFloat16: ADD SUB CMP MUL GETMAN* ADDIEXP* SUBEXP* EXP2F* F2INT* GETFR* INT2F* (*Used in complex ops)

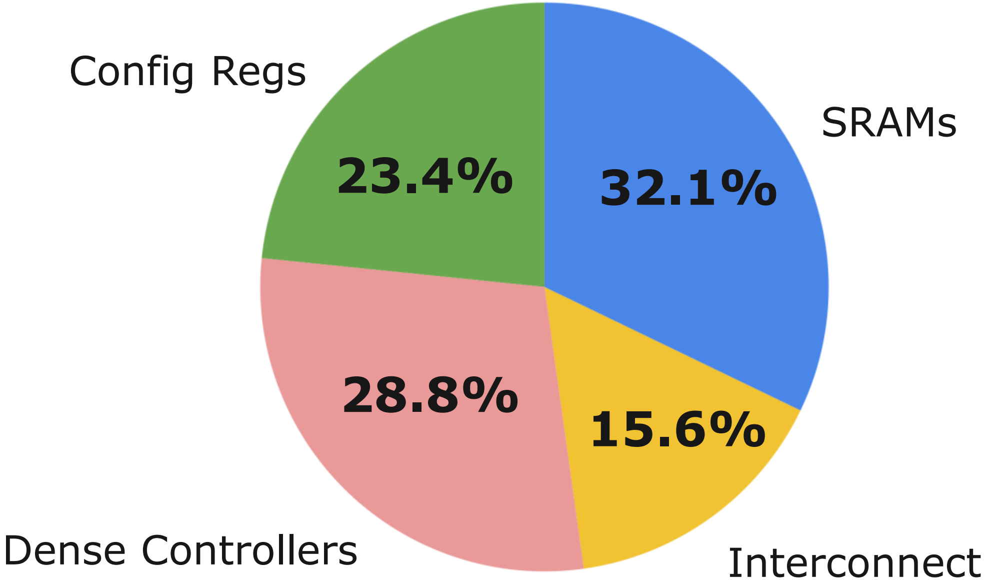
Optimized Compute Results

- Increased compute allows for further parallelization, reducing runtime

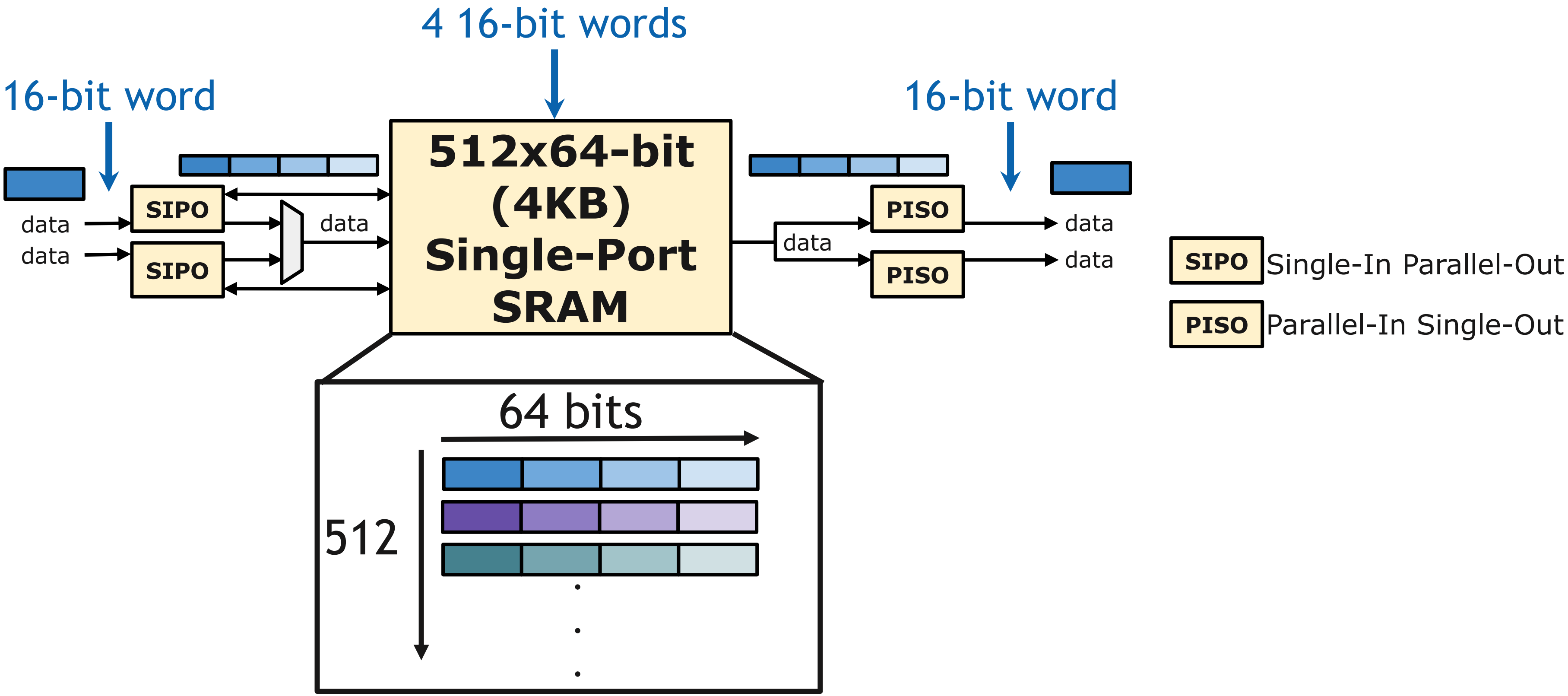


Challenge 2: Area Breakdown of MEM

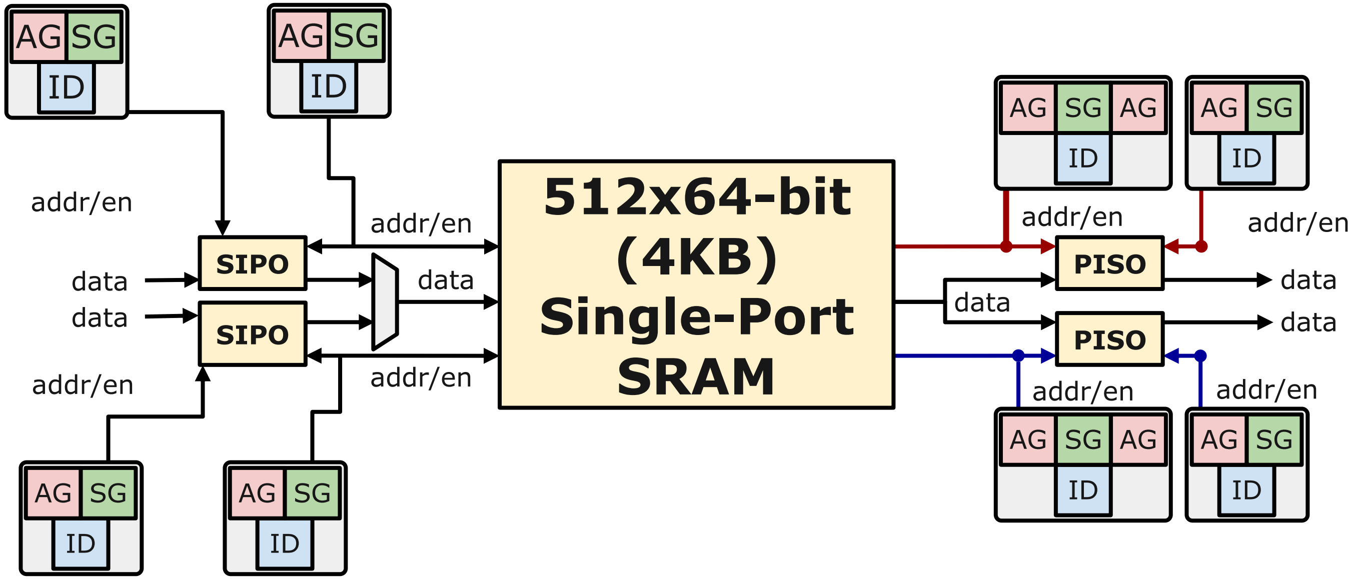
- SRAMs only 32.1% of MEM Tile



Dense Memory Controller


4 16-bit words
16-bit word
16-bit word
512x64-bit (4KB) Single-Port SRAM
SIPO
SIPO
data
data
data
PISO
PISO
data
data
data
SIPO Single-In Parallel-Out
PISO Parallel-In Single-Out
64 bits
512

Dense Memory Controller



AG Address Generator

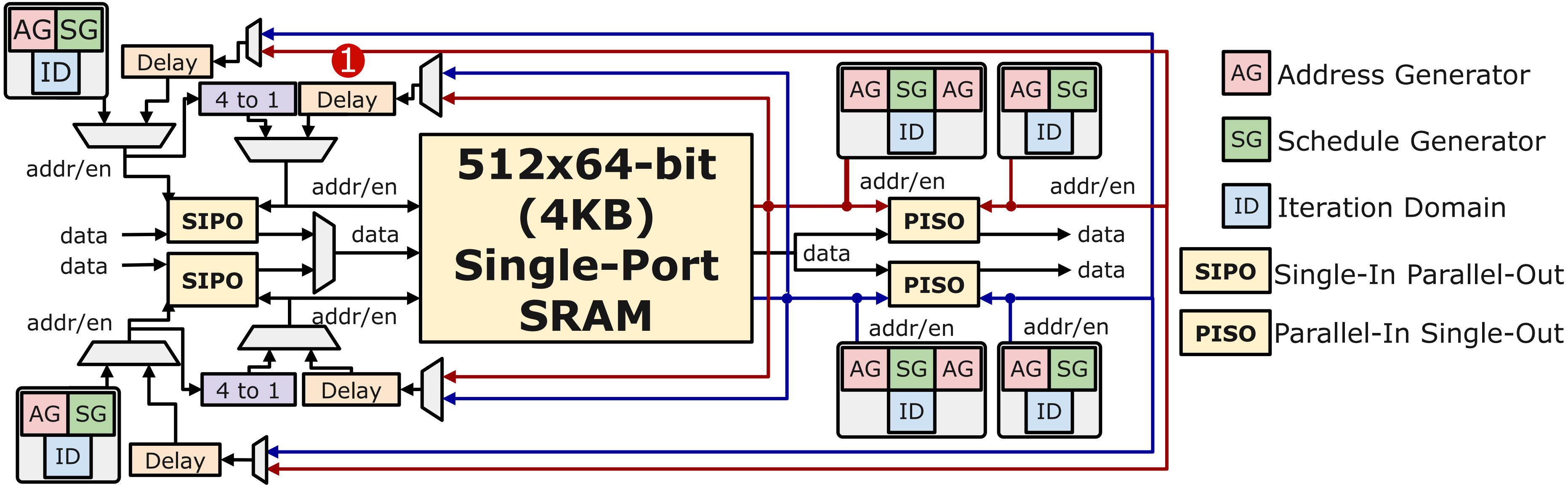
SG Schedule Generator

ID Iteration Domain

SIPO Single-In Parallel-Out

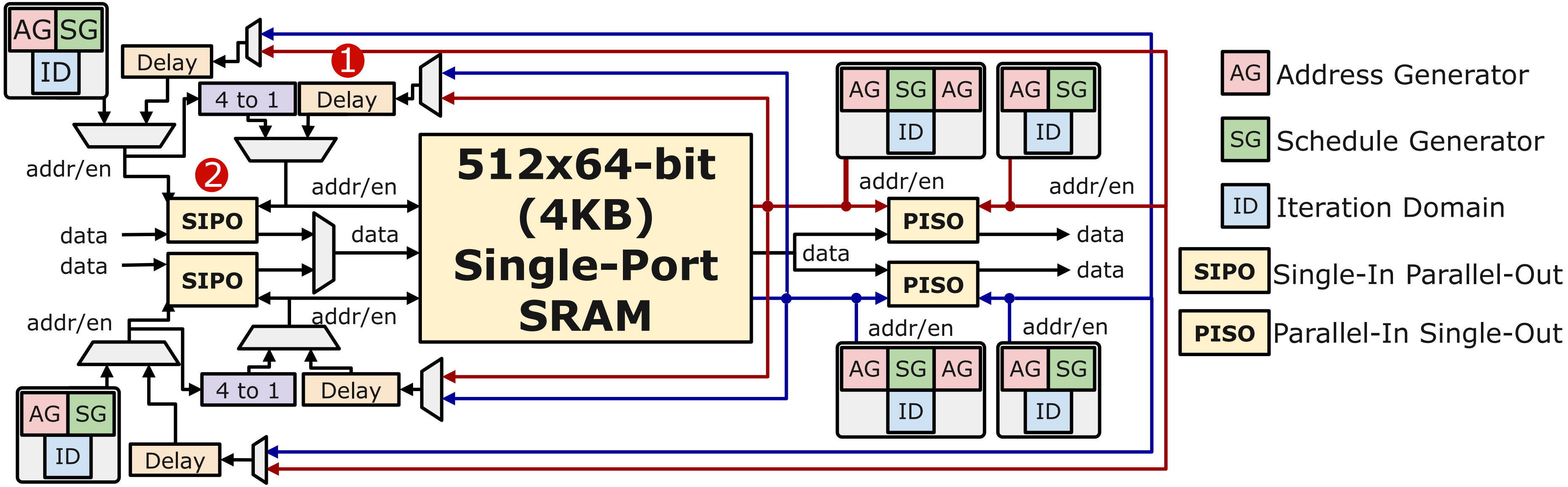
PISO Parallel-In Single-Out

Optimized Dense Memory Controller



1 Simplified write controllers by replacing address generators (AGs) with delay blocks for read-modify-write operations

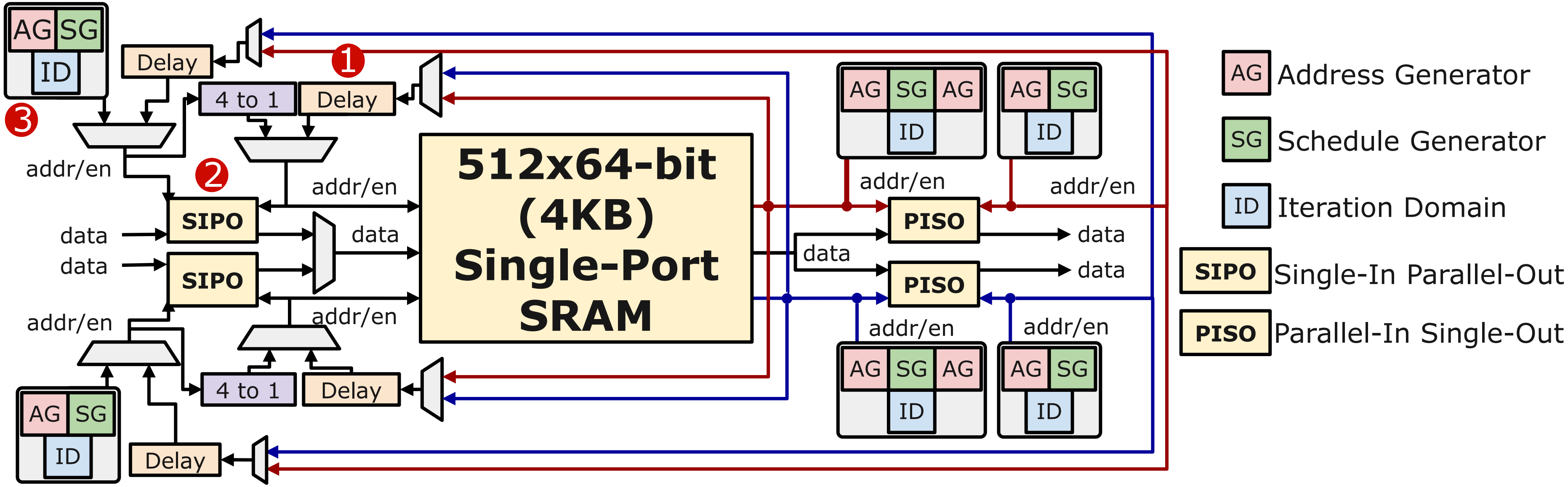
Optimized Dense Memory Controller



❶ Simplified write controllers by replacing address generators (AGs) with delay blocks for read-modify-write operations

❷ Halved SIPO FIFO depth

Optimized Dense Memory Controller



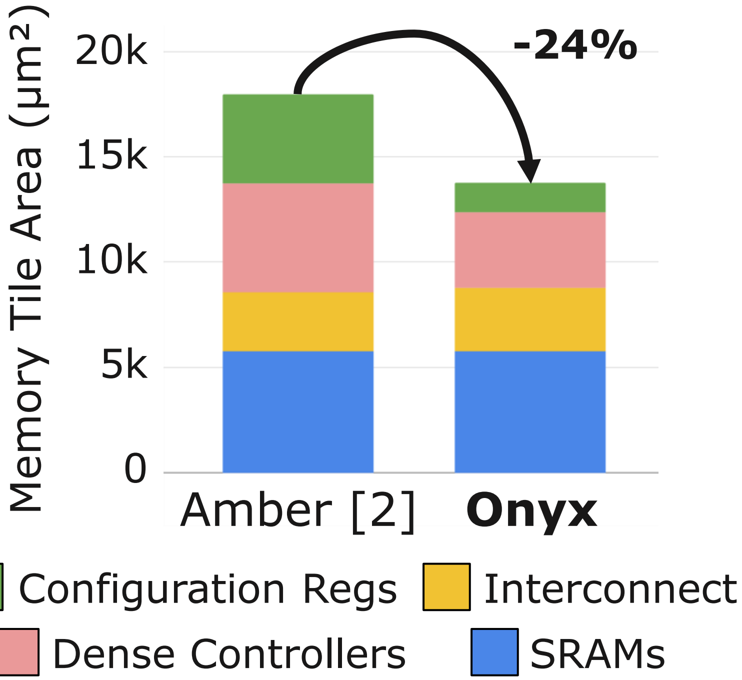
1 Simplified write controllers by replacing address generators (AGs) with delay blocks for read-modify-write operations

2 Halved SIPO FIFO depth

3 Reduced counter and configuration register bit widths in iteration domains (IDs)

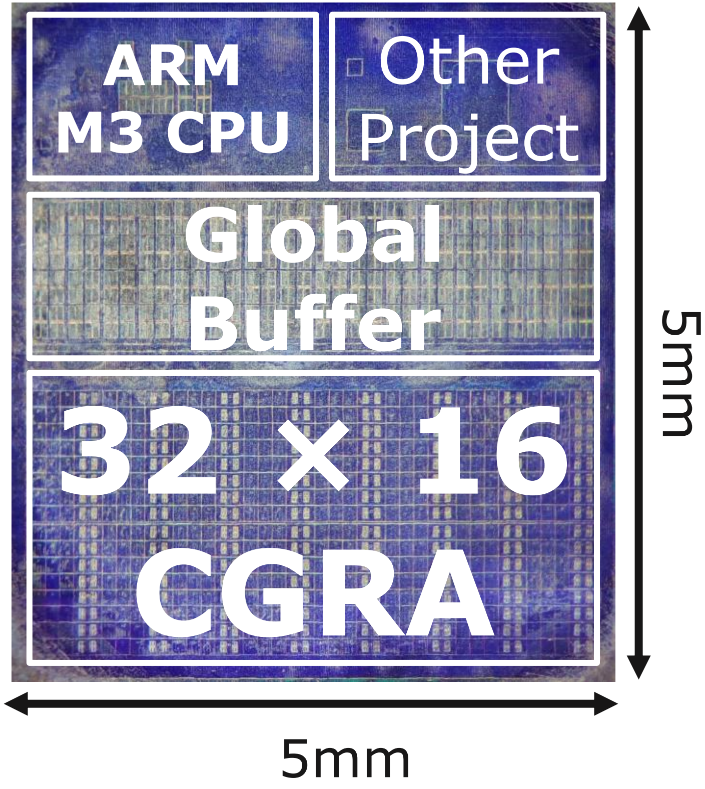
Dense Memory Controller Results

- Memory controller and configuration reduced by 24%



[2] K. Feng et al., JSSC'2023.

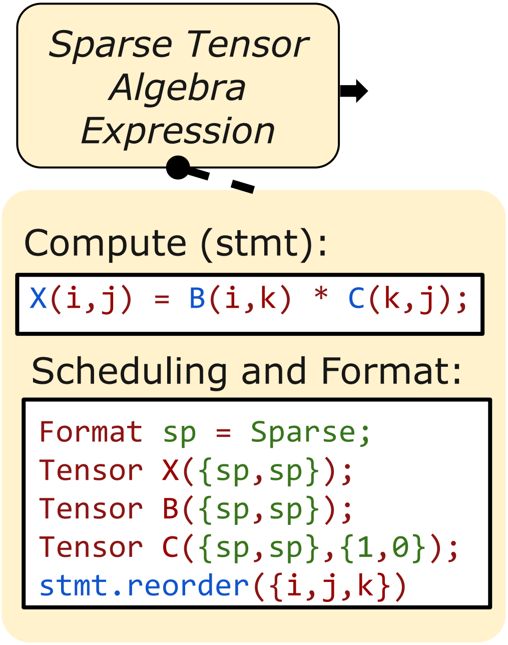
Chip Specifications and Die Photo



	Onyx
Architecture	SoC with CGRA
Node	GF 12 nm
Area (mm^2)	23
Precision	BF16, INT16
SRAM (MiB)	4.5
Voltage (V)	0.78
Freq (MHz)	970
Peak GOPS	571
GOPS/W	756

Application Compiler

- Designed end-end compiler to map Sparse Tensor Algebra Expressions onto CGRA



Sparse Tensor Algebra Expression

Compute (stmt):

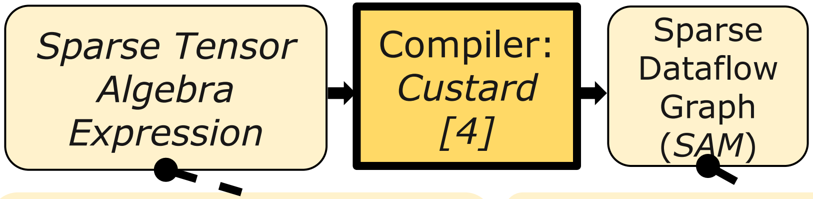
```
X(i,j) = B(i,k) * C(k,j);
```

Scheduling and Format:

```
Format sp = Sparse;
Tensor X({sp,sp});
Tensor B({sp,sp});
Tensor C({sp,sp},{1,0});
stmt.reorder({i,j,k})
```

Application Compiler

- Designed end-end compiler to map Sparse Tensor Algebra Expressions onto CGRA



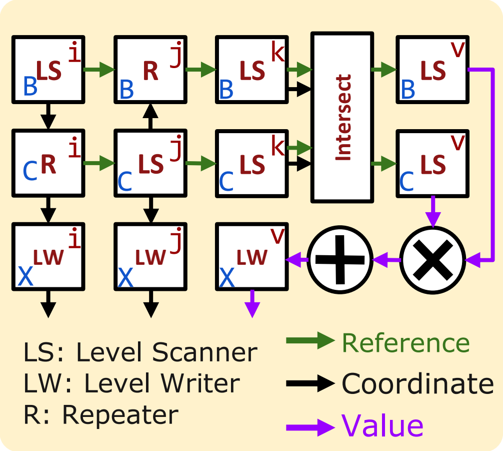
Sparse Tensor Algebra Expression → Compiler: *Custard [4]* → Sparse Dataflow Graph (*SAM*)

Compute (stmt):

```
X(i,j) = B(i,k) * C(k,j);
```

Scheduling and Format:

```
Format sp = Sparse;
Tensor X({sp,sp});
Tensor B({sp,sp});
Tensor C({sp,sp},{1,0});
stmt.reorder({i,j,k})
```



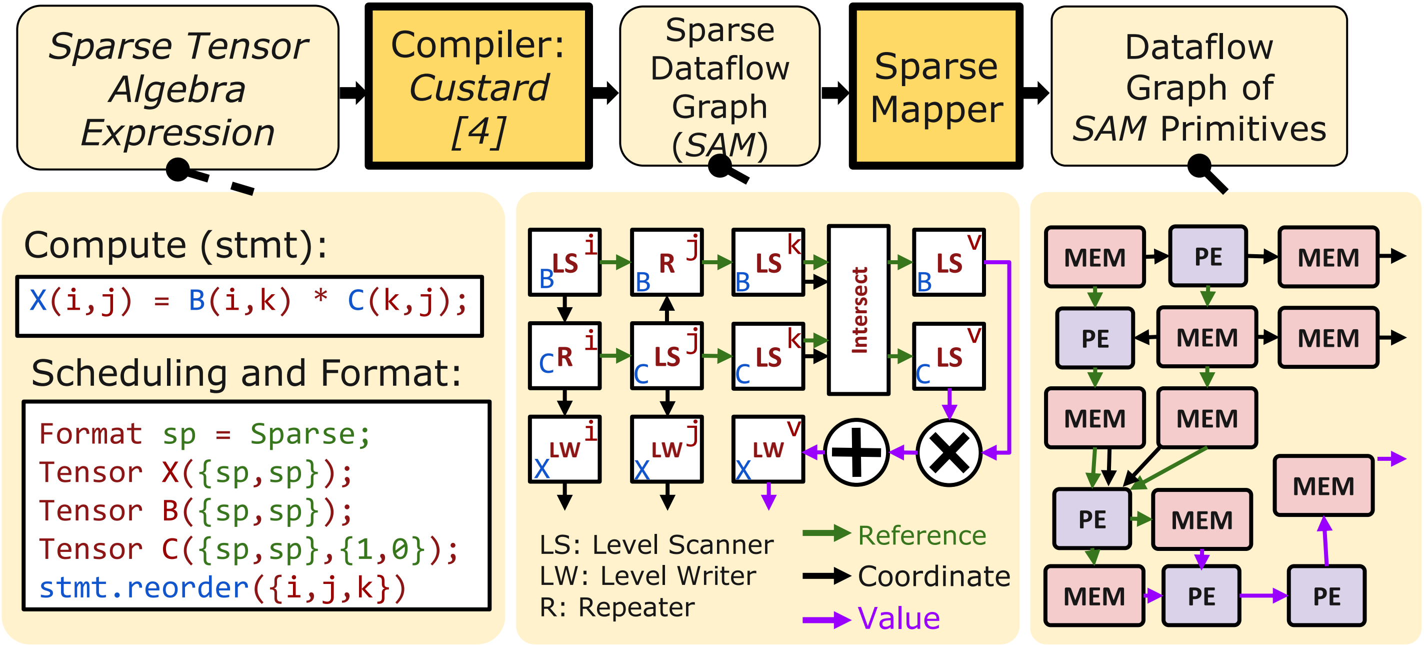
LS: Level Scanner
LW: Level Writer
R: Repeater

→ Reference
→ Coordinate
→ Value

[4] O. Hsu et al., ASPLOS’2023.

Application Compiler

- Designed end-end compiler to map Sparse Tensor Algebra Expressions onto CGRA



Compute (stmt):

```
X(i,j) = B(i,k) * C(k,j);
```

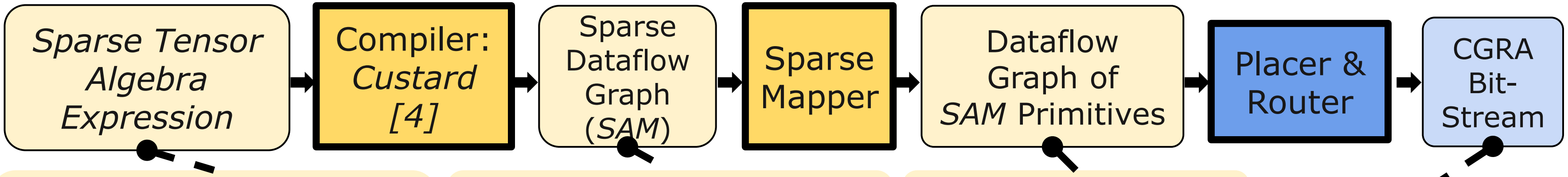
Scheduling and Format:

```
Format sp = Sparse;
Tensor X({sp,sp});
Tensor B({sp,sp});
Tensor C({sp,sp},{1,0});
stmt.reorder({i,j,k})
```

[4] O. Hsu et al., ASPLOS’2023.

Application Compiler

- Designed end-end compiler to map Sparse Tensor Algebra Expressions onto CGRA



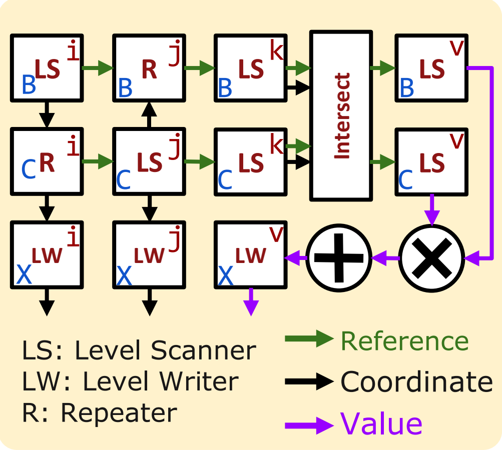
Sparse Tensor Algebra Expression → Compiler: *Custard [4]* → Sparse Dataflow Graph (*SAM*) → Sparse Mapper → Dataflow Graph of *SAM* Primitives → Placer & Router → CGRA Bit-Stream

Compute (stmt):

```
X(i,j) = B(i,k) * C(k,j);
```

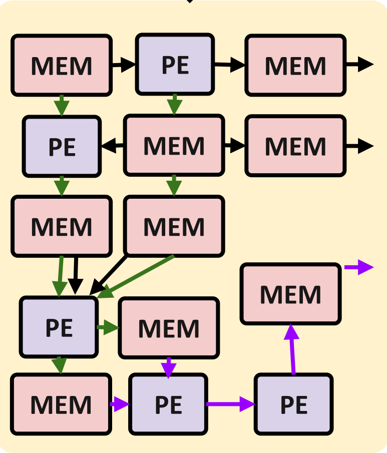
Scheduling and Format:

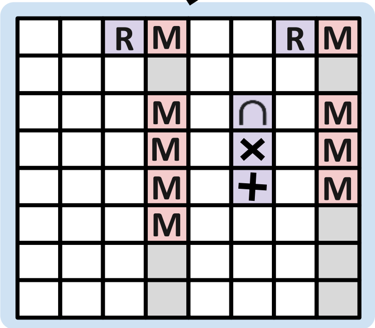
```
Format sp = Sparse;
Tensor X({sp,sp});
Tensor B({sp,sp});
Tensor C({sp,sp},{1,0});
stmt.reorder({i,j,k})
```



LS: Level Scanner
LW: Level Writer
R: Repeater

Reference
Coordinate
Value





[4] O. Hsu et al., ASPLOS’2023.

Benchmark Application Suite

Dense applications written in Halide compared against CPU, GPU, FPGA, and CGRA:

- Image processing
  - Blur: image blur
  - Unsharp: enhances local contrast by smoothing an image
  - Camera pipeline: processes raw data from an image sensor into a color image
- Computer vision
  - Harris: detects corners
- Machine learning
  - ResNet-18: image classification

Benchmark Application Suite

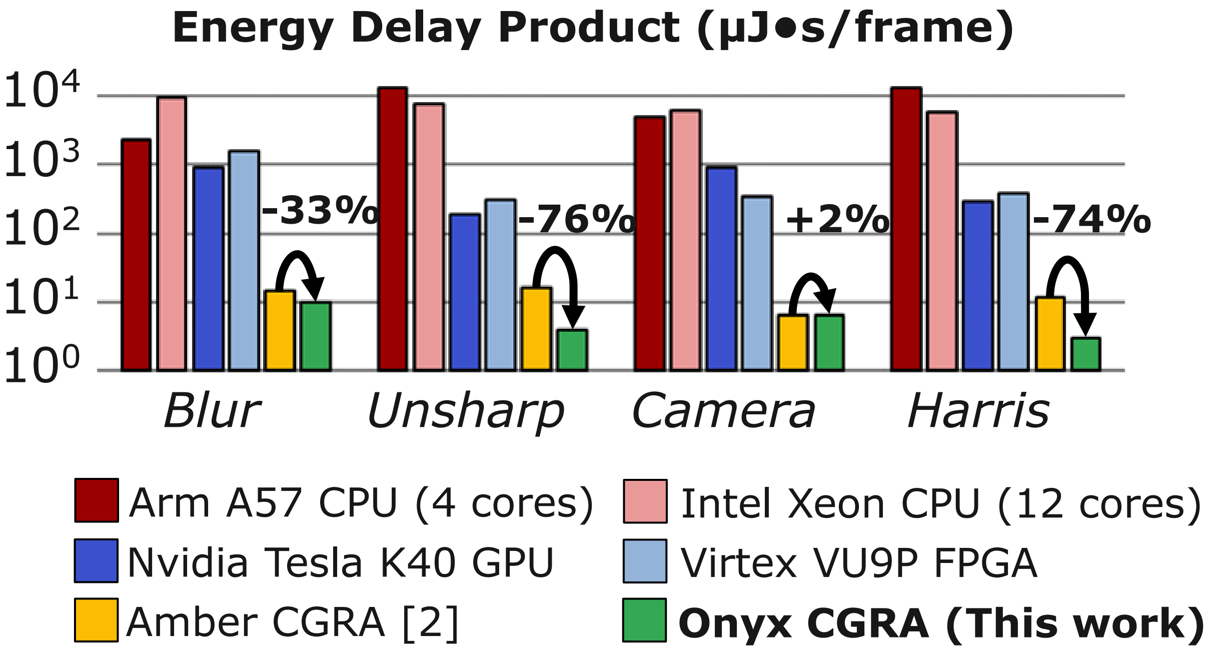
Sparse tensor algebra expressions compiled using Custard compared against CPU:

- 2D Dataset: Randomly selected SuiteSparse Matrices from linear programming, computational fluid dynamics, chemical process simulation, undirected weighted random graphs, etc.
  - Matrix Dimensions: 821x1876 - 2021x2021

- 3D Dataset: Uniform random generated sparse (67%) tensors
  - Tensors Dimensions: 8x37x10 - 28x35x54

Dense Image Processing Results

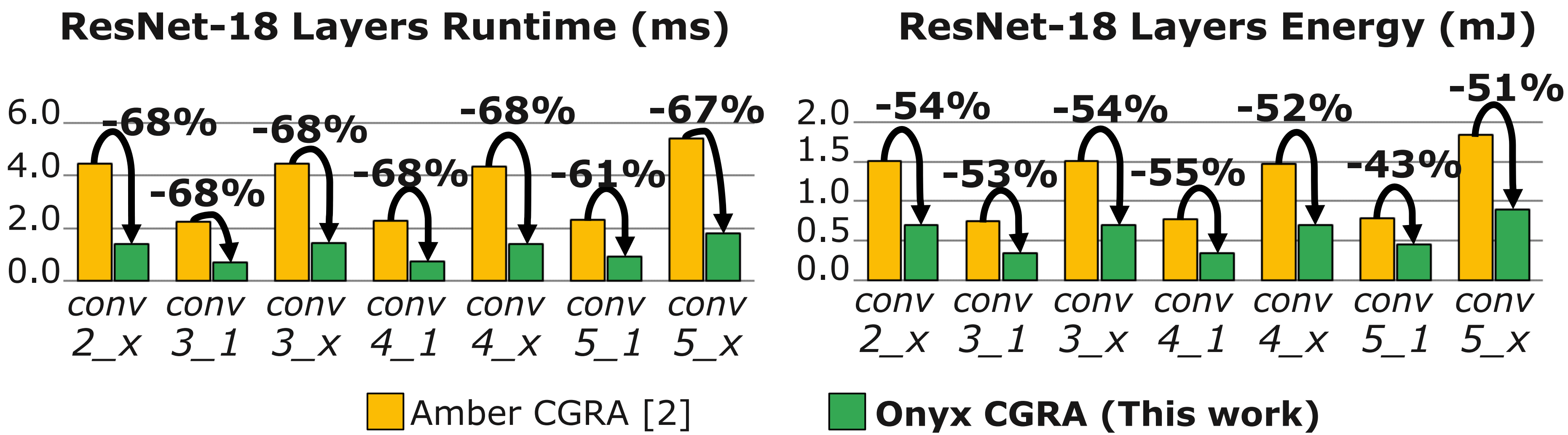
- Up to 76% better EDP versus the state-of-the-art CGRA



[2] K. Feng et al., JSSC'2023.

Dense Image ResNet Results

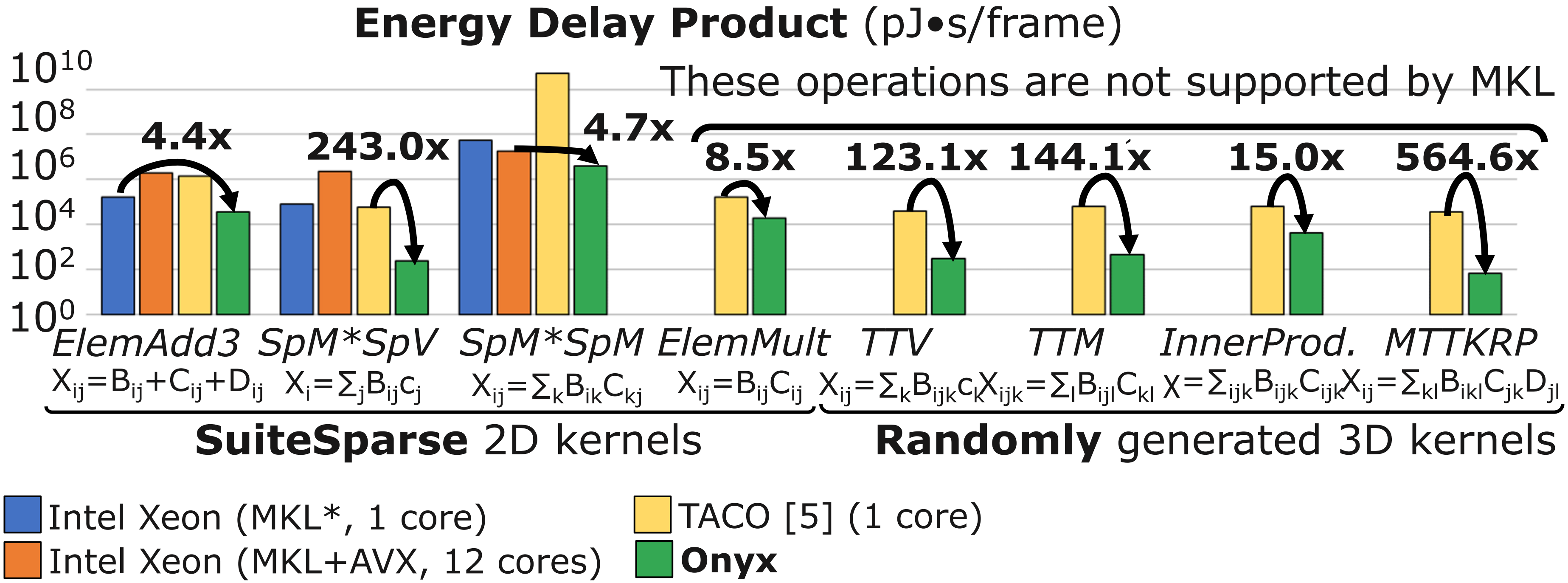
- Up to 55% better energy versus the state-of-the-art CGRA



[2] K. Feng et al., JSSC'2023.

Sparse Tensor Algebra Results

- Up to 564.6x better EDP versus a CPU



[5] F. Kjolstad et al., OOPSLA'2017.

Summary of Key Contributions

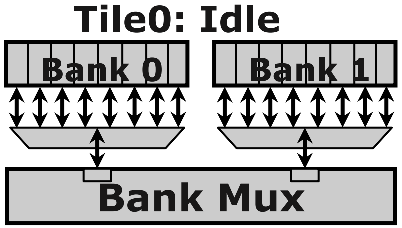
- Onyx is an SoC designed for flexible and efficient acceleration of sparse and dense data applications
  - **Sparse**: composable primitives for arbitrary sparse tensor algebra supporting multi-dimensional, multi-input expressions with fusion
  - **Dense**: compute density increased, and memory controllers optimized
- Automatic end-to-end compiler maps applications onto Onyx
- Onyx achieves up to 565x EDP improvement over CPUs with sparse libraries and 85% lower EDP versus the state-of-the-art CGRA on dense applications
- Demonstrates an approach for accelerating fast evolving application domains

Funding Acknowledgement: DARPA DSSoC, AHA Center, Stanford SystemX Alliance, SRC PRISM Center, NSF Award 2238006, Intel HIP and Samsung.

Backup Slides

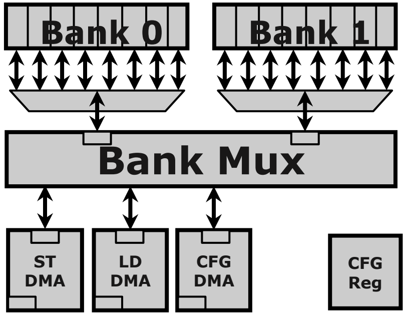
GLB Architecture

- Each GLB tile is composed of two banks



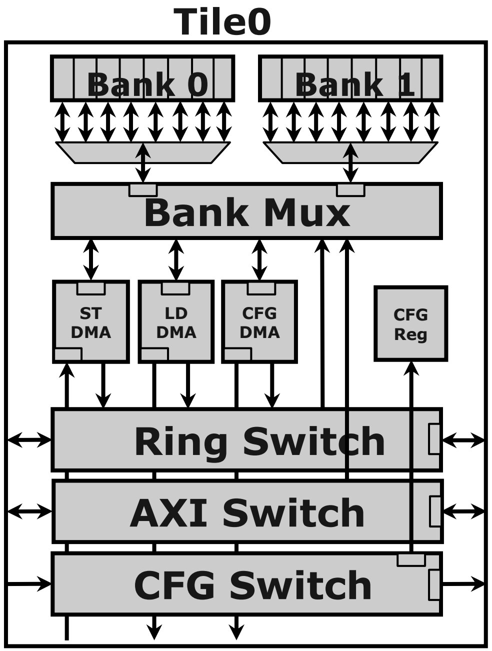
GLB Architecture

- Has Store, Load, Configuration DMAs and Configuration registers



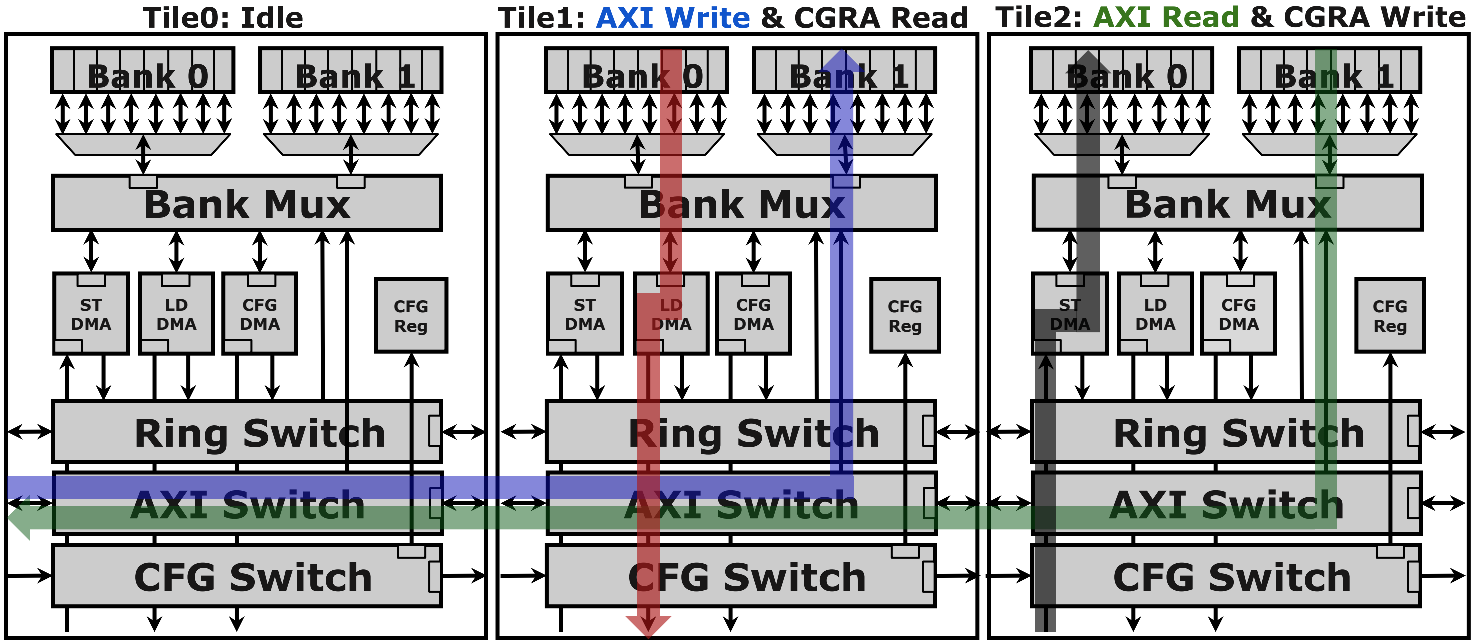
GLB Architecture

- Switches for CGRA and AXI communication and configuration



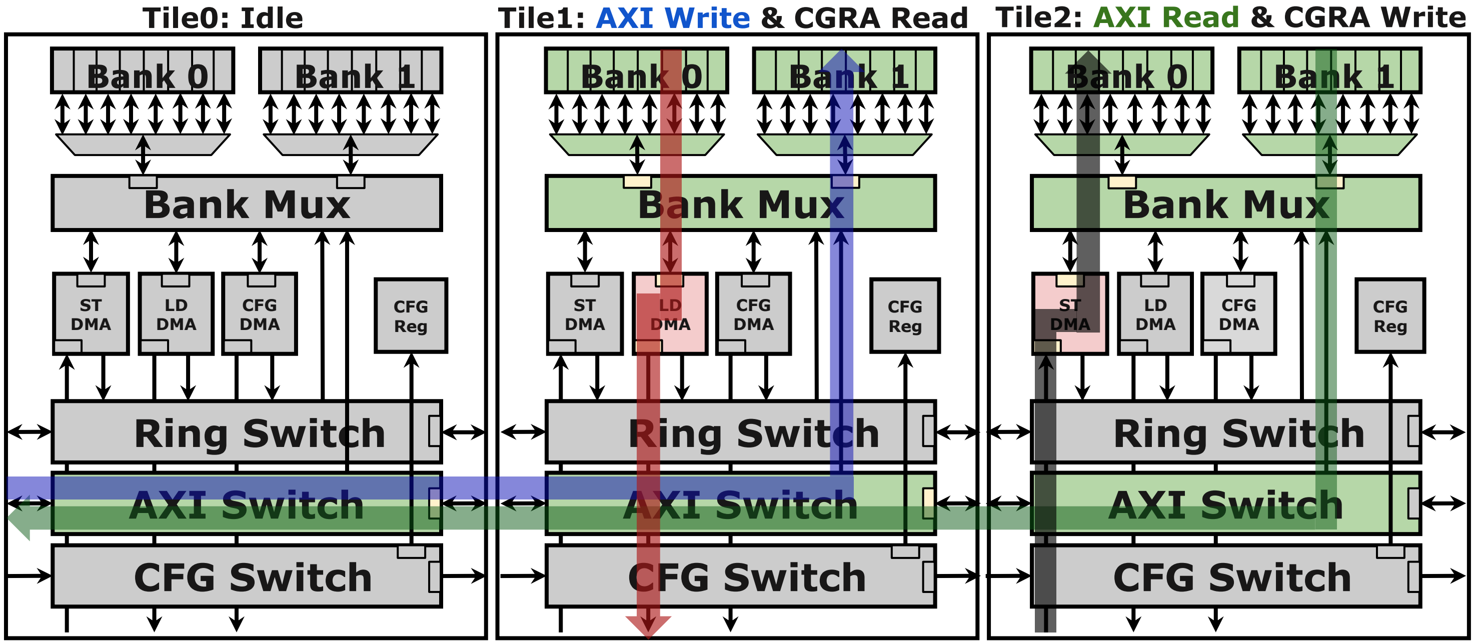
GLB Dynamic-/Static- Clock Gating

- Double buffering workload: what blocks are active?



GLB Dynamic-/Static- Clock Gating

- Only parts of the blocks are active during read/write transactions



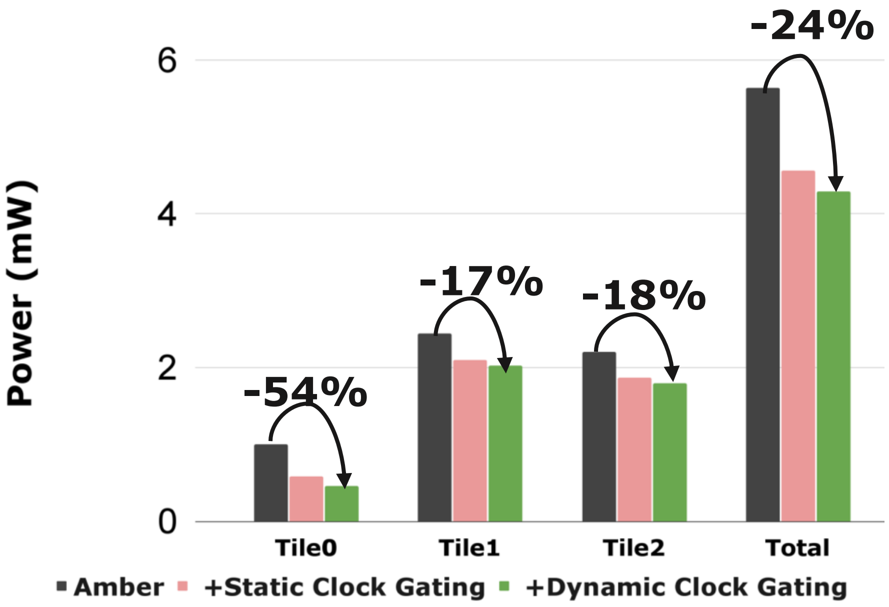
🟥 Static Clock Gating (SCG): Gating based on internal states

🟩 Dynamic Clock Gating (DCG): Gating based on read/write activity

⬜ Gated Block: Fully gated if not used in workload

GLB Results

- 24% power saved on example double buffering workload


Power (mW)
Tile0
Tile1
Tile2
Total
■ Amber
■ +Static Clock Gating
■ +Dynamic Clock Gating