Guest Lecture
StreamTensor: Make Tensors Stream in Dataflow Accelerators for LLMs

Hanchen Ye, ElastixAI
hanchenye@gmail.com
March 10, 2025

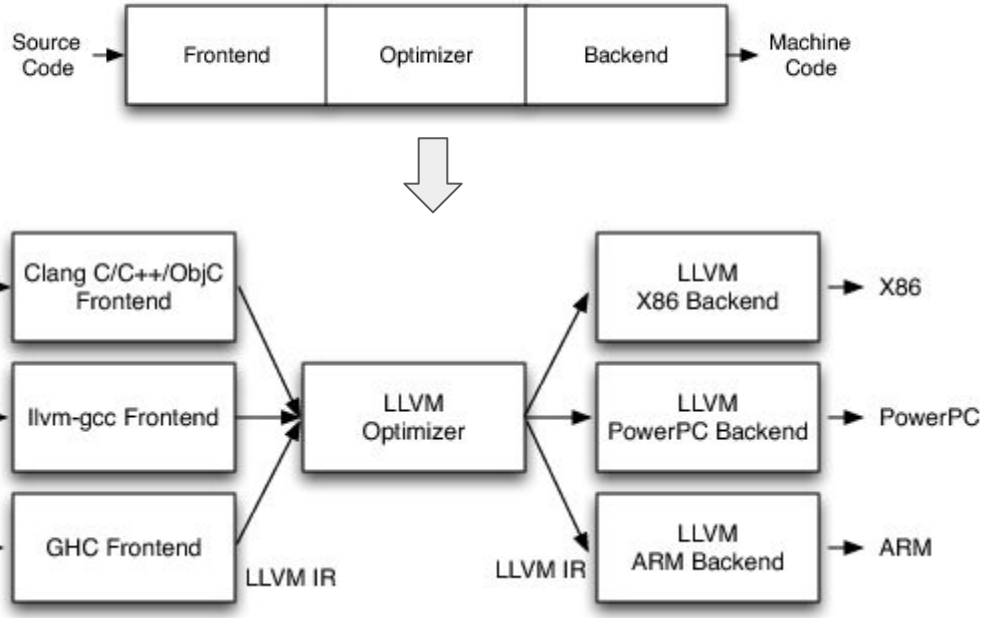
Outline

- Motivation
- StreamTensor Typing System
- StreamTensor Compilation Pipeline
- StreamTensor Design Spaces
- StreamTensor Results

StreamTensor Outline

- Motivation
- StreamTensor Typing System
- StreamTensor Compilation Pipeline
- StreamTensor Design Spaces
- StreamTensor Results

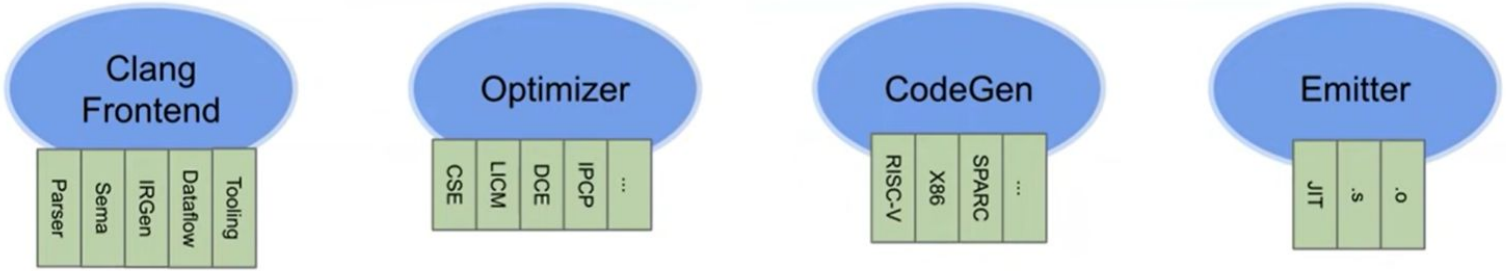
LLVM: Compiler Infrastructure



- LLVM uses the same **intermediate representation (IR)** to represent ALL programs.
- All program optimizations are based on the LLVM IR.
- LLVM dispatches the front-ends, optimizations, and back-ends. O(m*n) -> O(1)

Source: The architecture of open-source applications, C. Lattner.

LLVM: Compiler Infrastructure (Cont'd)



Key insight: Compilers as libraries, not an app!

- Enable embedding in other applications
- Mix and match components
- No hard coded lowering pipeline

Source: The Golden Age of Compiler Design in an Era of HW/SW Co-design, C. Lattner.

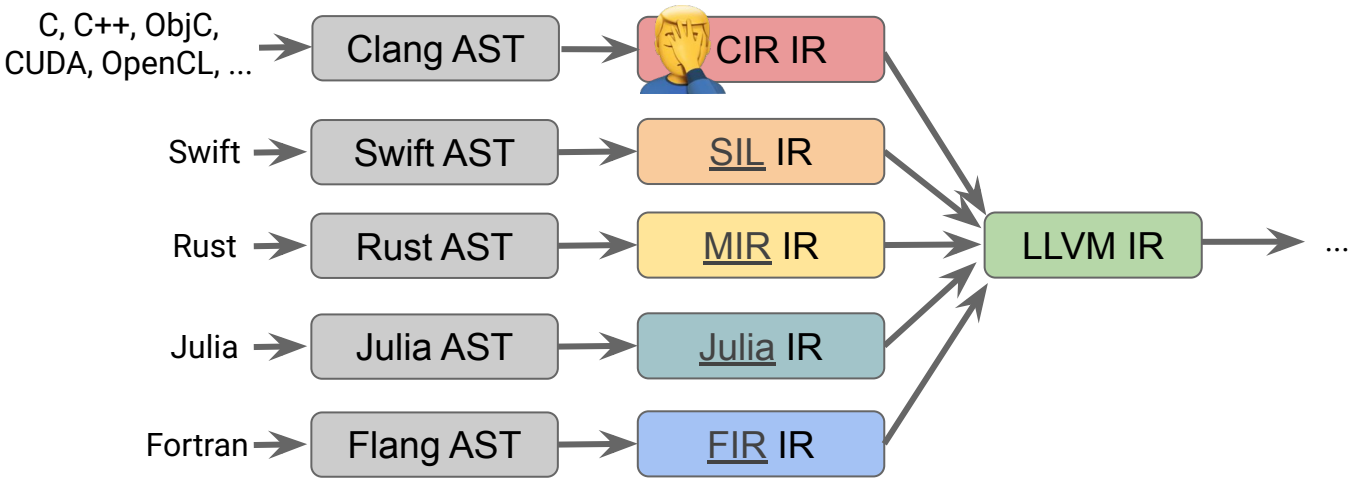
LLVM: Compiler Infrastructure (Cont'd)

What is LLVM?

- An open source framework for building tools
  - Tools are created by linking together various libraries provided by the LLVM project and your own
- An extensible, strongly typed intermediate representation, i.e. LLVM IR
  - https://llvm.org/docs/LangRef.html
- An industrial strength C/C++ optimizing compiler
  - Which you might know as clang/clang++ but these are really just drivers that invoke different parts (libraries) of LLVM

Source: Introduction to LLVM, UG3 Compiling Techniques.

From LLVM to MLIR



- More and more programming languages demand customized IR for optimization.
- The IR for different languages have different abstraction level.
- Language-specific IR can be lowered to LLVM for back-end code generation.

Source: MLIR: Multi-Level Intermediate Representation Compiler Infrastructure, C. Lattner.

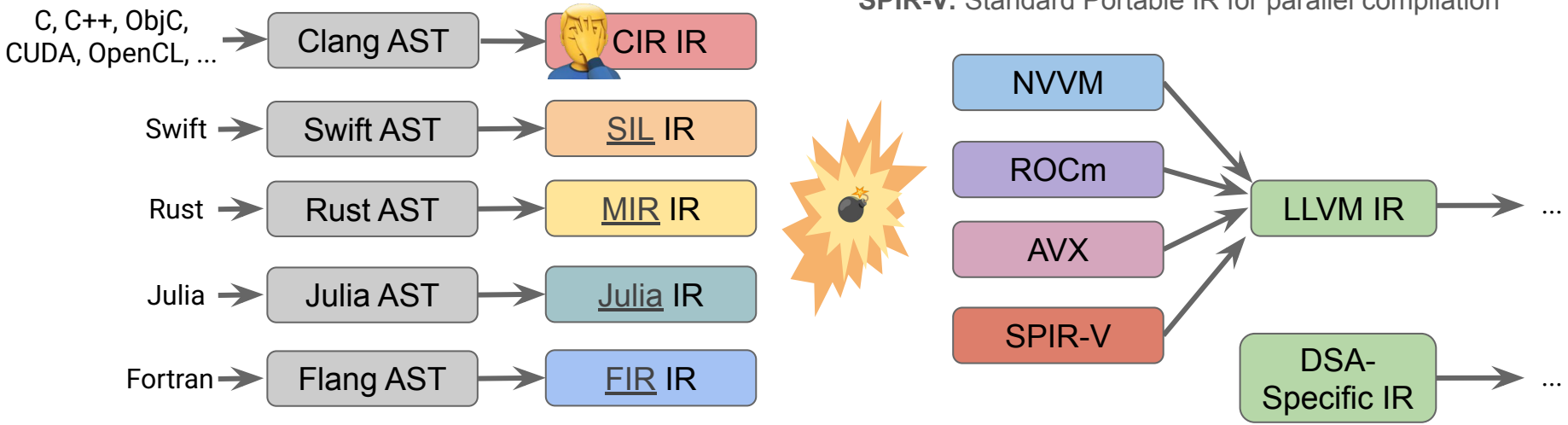
From LLVM to MLIR (Cont'd)

NVVM: IR for Nvidia GPU

ROCm: IR for AMD GPU

AVX: IR for Intel vector extension

SPIR-V: Standard Portable IR for parallel compilation



- Different back-ends demand customized IR for optimization
- DSAs (Domain-Specific Accelerator) even cannot use LLVM for generating back-end codes and demand their own IR for code generation

Severe Fragmentation: IRs have different implementations and "frameworks"

MLIR: Compiler Infrastructure for the End of Moore's Law

- **M**ulti-**L**evel **I**ntermediate **R**epresentation
- State of the art compiler technology
- Built on top of LLVM's open and library-based philosophy
- **Modular and extensible**
- Originally created within Google for compiling TensorFlow
- **Sufficiently general** to compile lots of domains

https://mlir.llvm.org

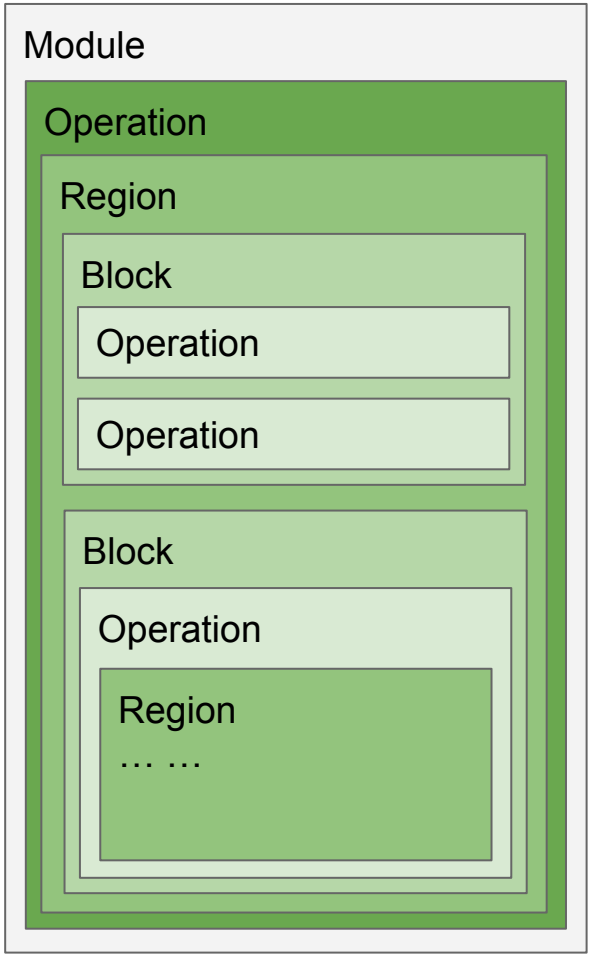
Source: Applying Circuit IR Compilers and Tools (CIRCT) to ML Applications, M. Urbach.

Syntax of MLIR

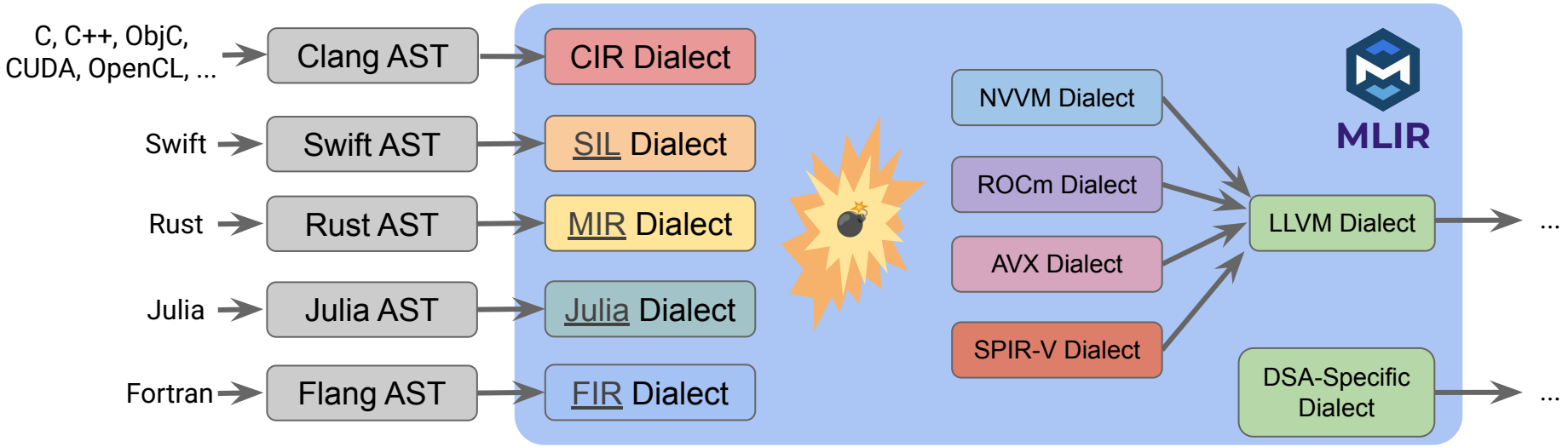
- SSA-based IR design, explicit typing system
- Module/Operation/Region/Block/Operation hierarchy
- Operation can contain multiple Regions

```
func.func @testFunction(%arg0: i32) -> i32 {
  %a = func.call @thingToCall(%arg0) : (i32) -> i32
  cf.br ^bb1
^bb1:
  %c = affine.for %i = 0 to 10 iter_args(%b = %a) -> i32 {
    %i_i32 = arith.index_cast %i : index to i32
    %b_new = arith.addi %i_i32, %b : i32
    affine.yield %b_new : i32
  }
  func.return %c : i32
}
```

Dialect A C++ namespace that contains customized operations, types, and attributes. Implement the “correct” abstraction for your domain.



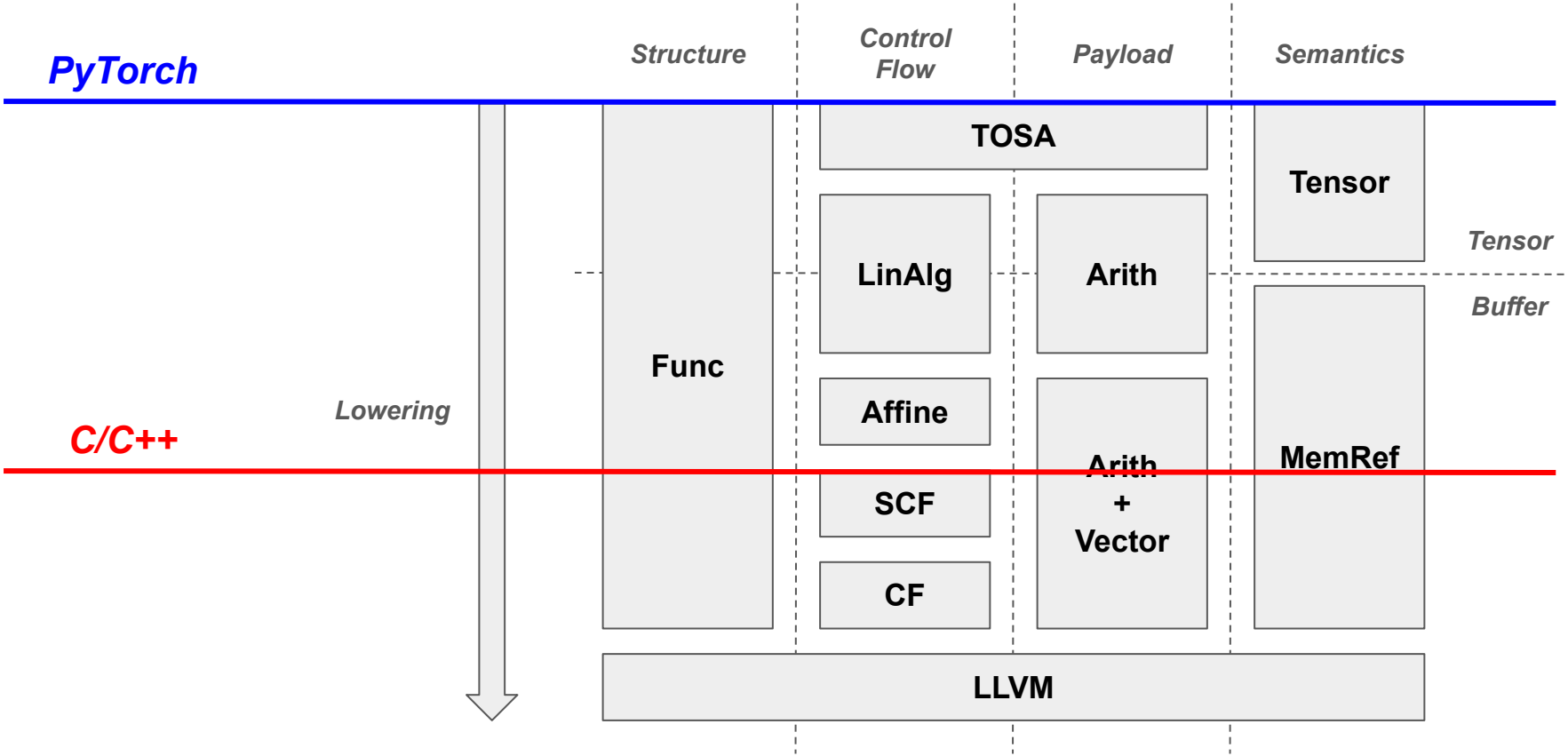
MLIR: “Meta IR” and Compiler Infrastructure



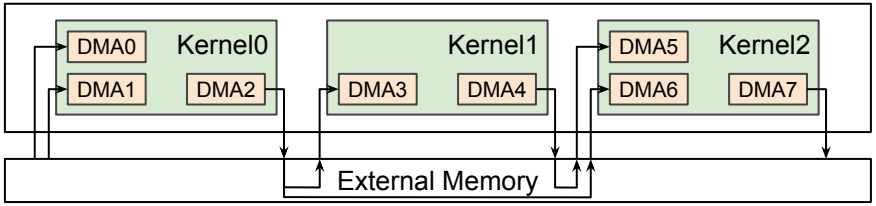
MLIR is a **“Meta IR”** and **compiler infrastructure** for:

- Design and implement **dialect**
- Optimization and transform inside of a **dialect**
- Conversion between different **dialects**
- Code generation of **dialect**

MLIR: “Meta IR” and Compiler Infrastructure (Cont’d)



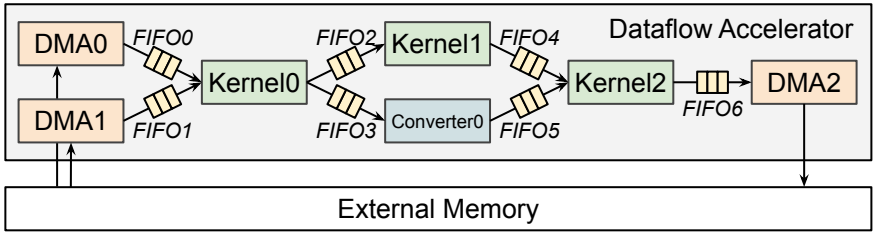
Motivation – Fight with the Memory Wall



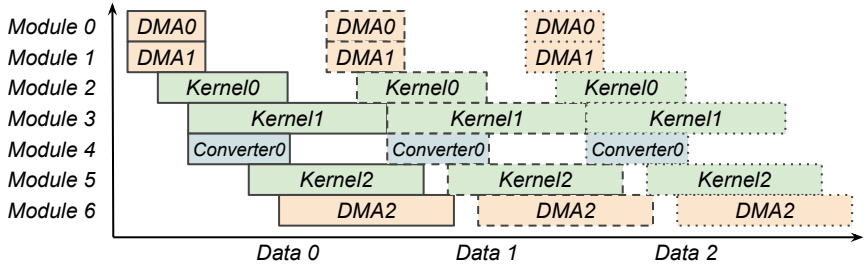
(a) A buffer-based dataflow accelerator

- Intermediate results are communicated through **on-chip ping-pong buffers**
- If on-chip memory resources are not enough, external memory is used
- Lead to frequent external memory access

Stream-based Kernel Fusion



(b) A stream-based dataflow accelerator



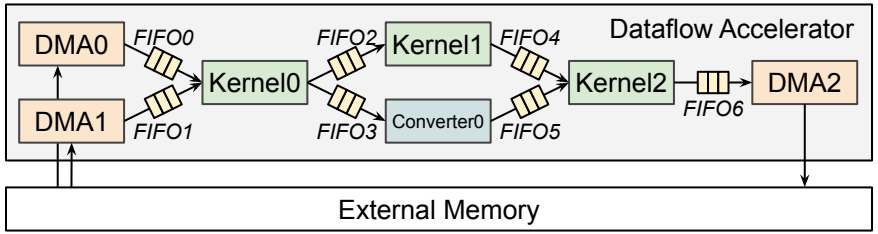
(c) Schedule of the stream-based dataflow accelerator

Reduce on-chip buffer utilization ✅

Reduce external memory access ✅

Reduce overall latency & throughput ✅

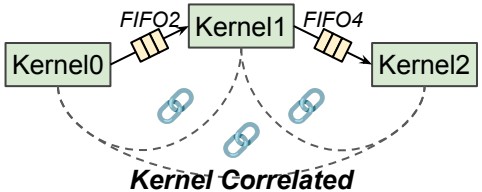
Pitfalls of Stream-based Dataflow Accelerator



Pitfall 1 ❎ External Memory Access

- Best external memory layout matching stream pattern?
- Widen external memory interfaces to maximize bandwidth?

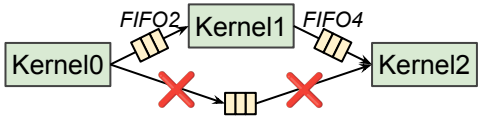
Pitfall 2 ❎
Inter-kernel Correlation



Kernel Correlated

- Kernel tiling & parallelization?
- Local buffer partition?
- Kernel loop permutation?

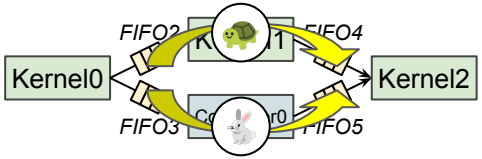
Pitfall 3 ❎
Kernel Fusion



Stream Layout/Order Mismatch

- Possible to stream?
- Minimal stream buffer size?
- Global fusion strategy?

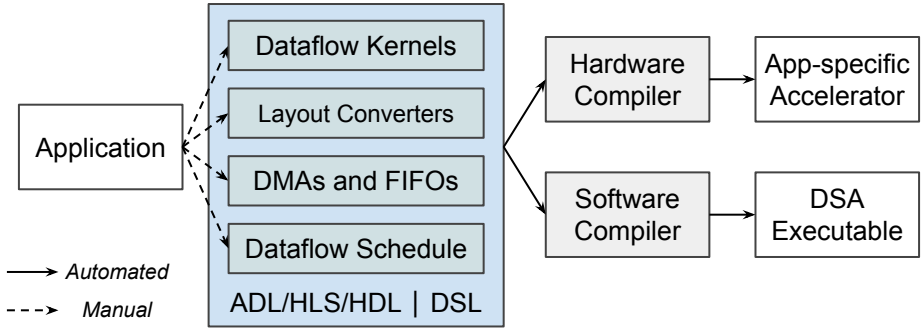
Pitfall 4 ❎
FIFO Sizing



Stream Deadlock

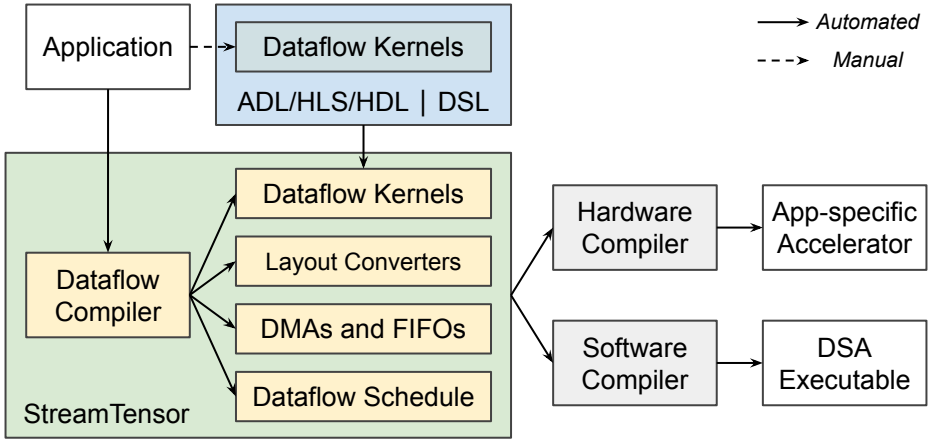
- Latency mismatch on different paths between two kernels
- Minimal FIFO size to avoid deadlock or kernel stall?

Dataflow Accelerator Design Paradigm



- Manual dataflow kernels, layout converters, DMAs, and FIFOs design
- Difficult to comprehend the optimal solutions of the pitfalls
- Low design productivity

Paradigm Shift

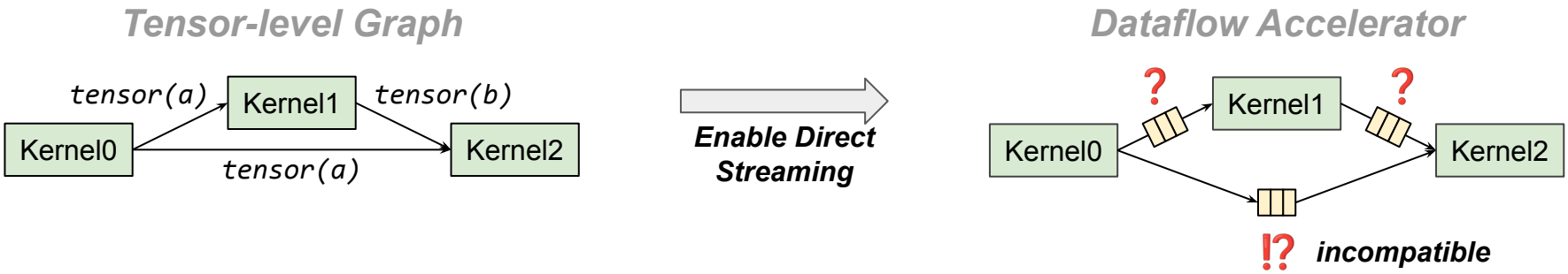


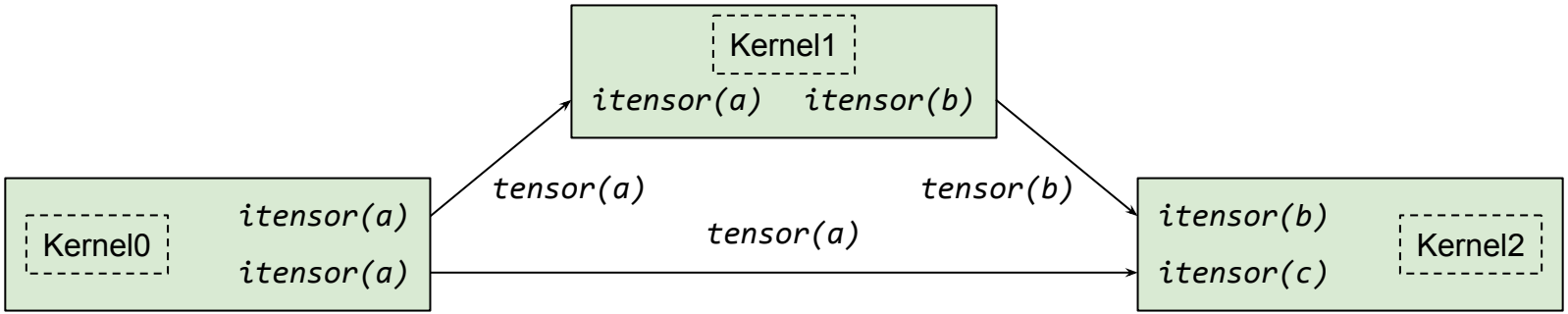
- Automate the generation of layout converters, DMAs, and FIFOs
- Resolve pitfalls through systematic design space exploration
- Support auto-tuned or hand-written kernel integration

StreamTensor Outline

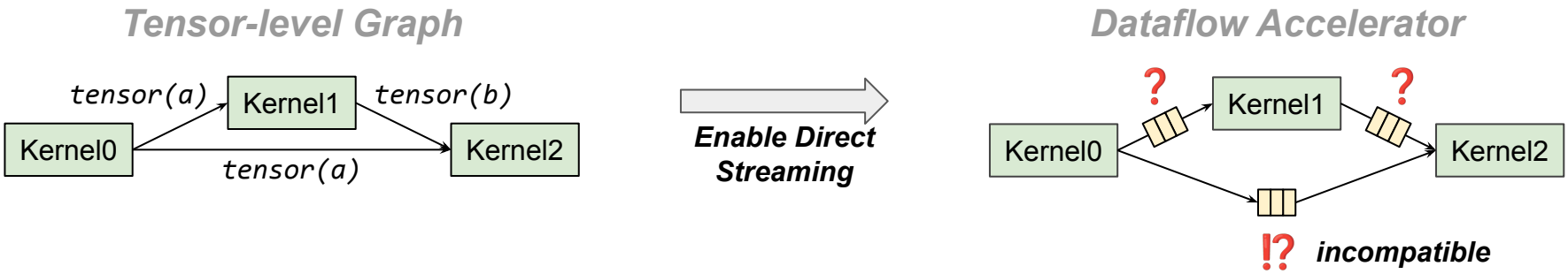
- Motivation
- StreamTensor Typing System
- StreamTensor Compilation Pipeline
- StreamTensor Design Spaces
- StreamTensor Results

Motivation of Iterative Tensor Type

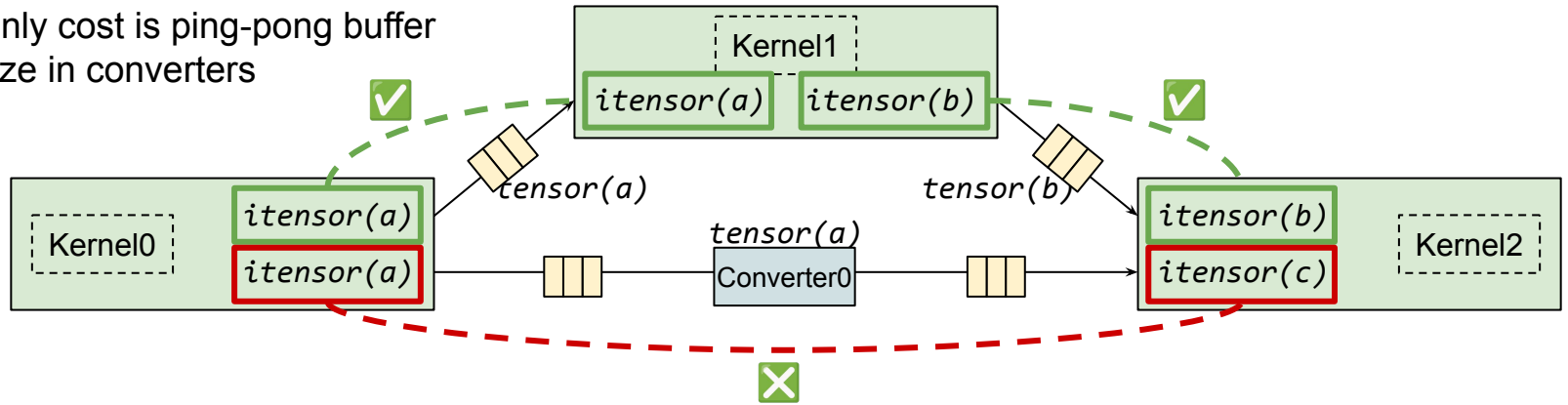




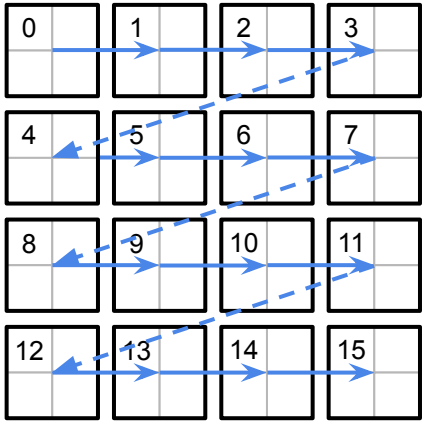
Motivation of Iterative Tensor Type (Cont.)



- By design, all kernels can be fused in a streaming manner
- Only cost is ping-pong buffer size in converters



Iterative Tensor Type



(a) itensor<2x2xf32,
iter_space: [4,4]*[2,2],
iter_map: (d0,d1)->(d0,d1)>

2x2xf32 = Element Shape & Type
iter_space = Iteration Space
= [Tripcounts]*[Steps]
iter_map = Iteration Affine Map

Iteration Space
[4,4]*[2,2]

[0,0]
[0,2]
[0,4]
[0,6]
[2,0]
[2,2]
…

Affine Mapping
(d0,d1)->(d0,d1)

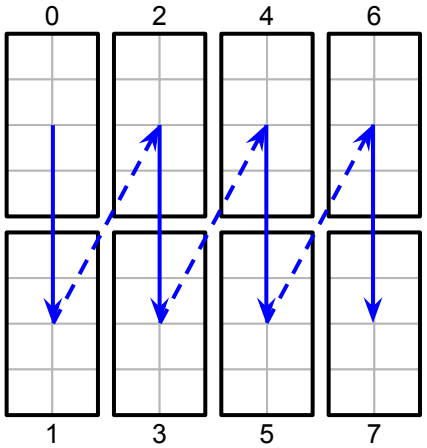
Element Shape
2x2

Data Space

[0,0]
[0,2]
[0,4]
[0,6]
[2,0]
[2,2]
…

Same `itensor` type means same stream layout/order

Iterative Tensor Type (Cont.)



(b) `itensor``<4x2xf32, iter_space: [4,2]*[2,4], iter_map: (d0,d1)->(d1,d0)>`

`4x2xf32` = Element Shape & Type

`iter_space` = Iteration Space
= [Tripcounts]*[Steps]

`iter_map` = Iteration Affine Map

Iteration Space
`[4,2]*[2,4]`

`[0,0]`
`[0,4]`
`[2,0]`
`[2,4]`
`[4,0]`
`[4,4]`
…

Affine Mapping
`(d0,d1)->(d1,d0)`

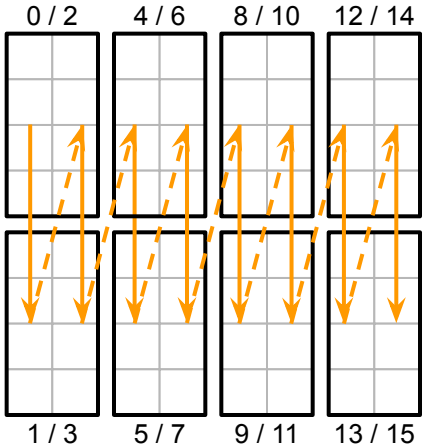
Element Shape
4x2

Data Space

`[0,0]`
`[4,0]`
`[0,2]`
`[4,2]`
`[0,4]`
`[4,4]`
…

Transposed stream layout/order

Iterative Tensor Type (Cont.)



(c) `itensor``<4x2xf32, iter_space: [4,2,2]*[2,1,4], iter_map: (d0,d1,d2)->(d2,d0)>`

`4x2xf32` = Element Shape & Type

`iter_space` = Iteration Space
= [Tripcounts]*[Steps]

`iter_map` = Iteration Affine Map

Iteration Space
`[4,2,2]*[2,1,4]`

`[0,0,0]`
`[0,0,4]`
`[0,1,0]`
`[0,1,4]`
`[2,0,0]`
`[2,0,4]`
…

Affine Mapping
`(d0,d1,d2)->(d2,d0)`

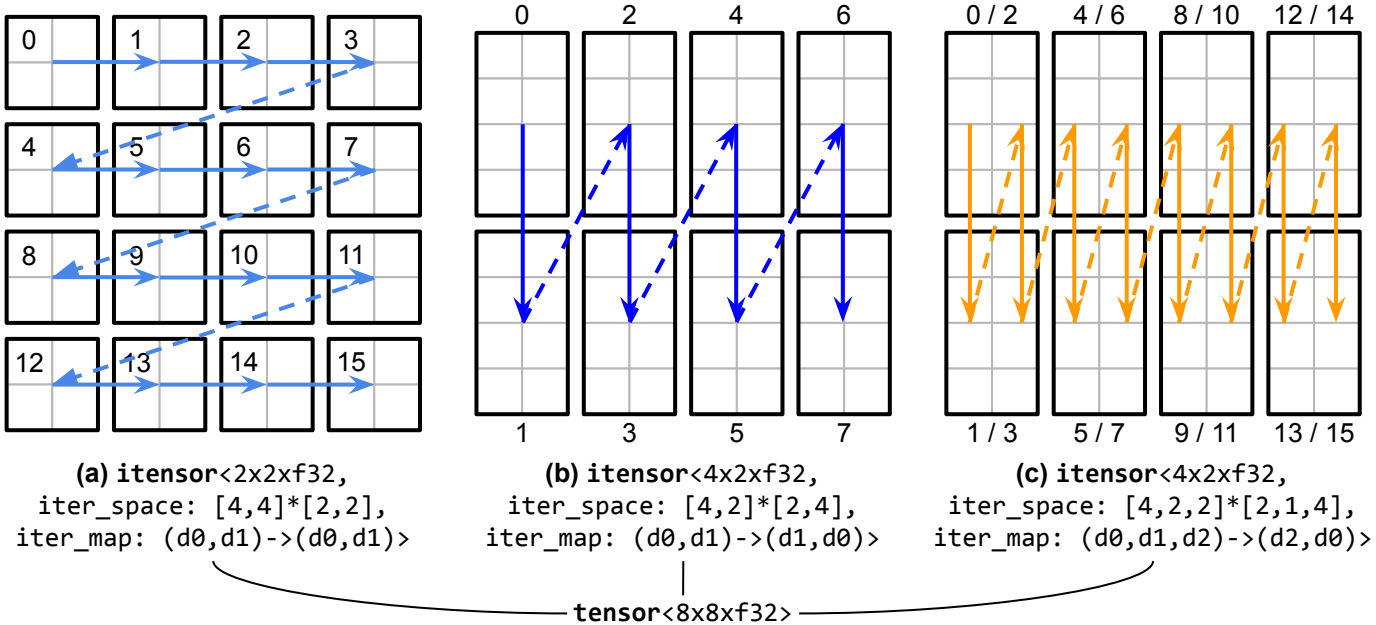
Element Shape
4x2

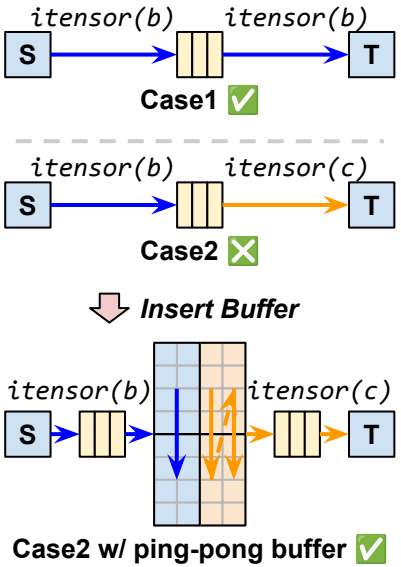
Data Space

`[0,0]`
`[4,0]`
`[0,0]`
`[4,0]`
`[0,2]`
`[4,2]`
…

Transposed and repeated stream layout/order

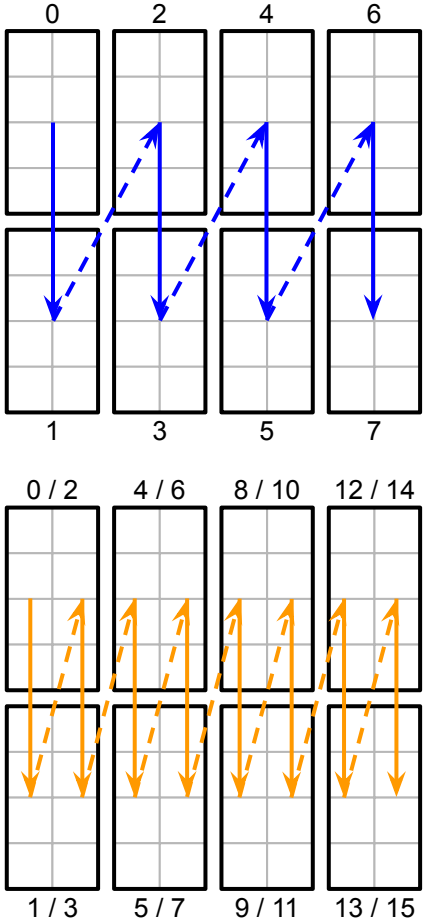
Iterative Tensor Type (Cont.)





The minimal buffer size is inferred from `itensor` types, which are used as "cost" during kernel fusion

Infer Buffer Shape from Iterative Tensor Types



(b) `itensor<4x2xf32,`

`iter_space: [4,2]*[2,4],`

`iter_map: (d0,d1)->(d1,d0)>`

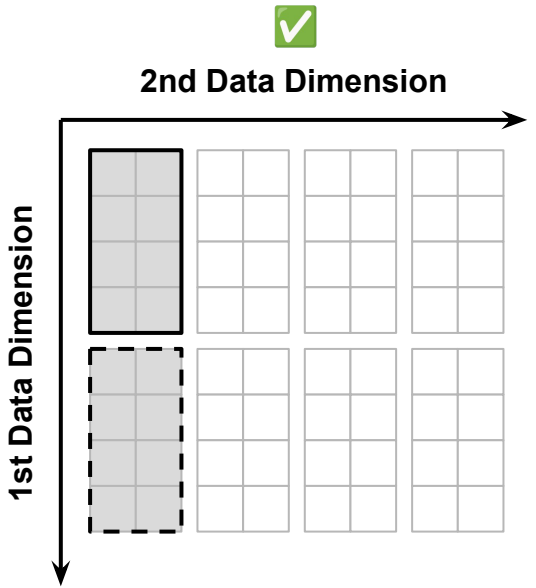
Element Shape ✅

Iteration Dimension ✅

(c) `itensor<4x2xf32,`

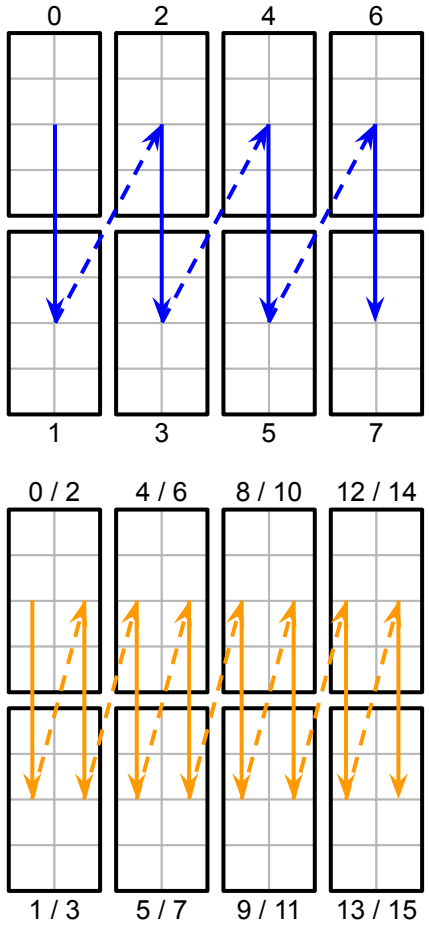
`iter_space: [4,2,2]*[2,1,4],`

`iter_map: (d0,d1,d2)->(d2,d0)>`



Traverse each data dimension from little-endian

Infer Buffer Shape from Iterative Tensor Types (Cont.)



(b) `itensor<4x2xf32,`

`iter_space: [4,2]*[2,4],`

`iter_map: (d0,d1)->(d1,d0)>`

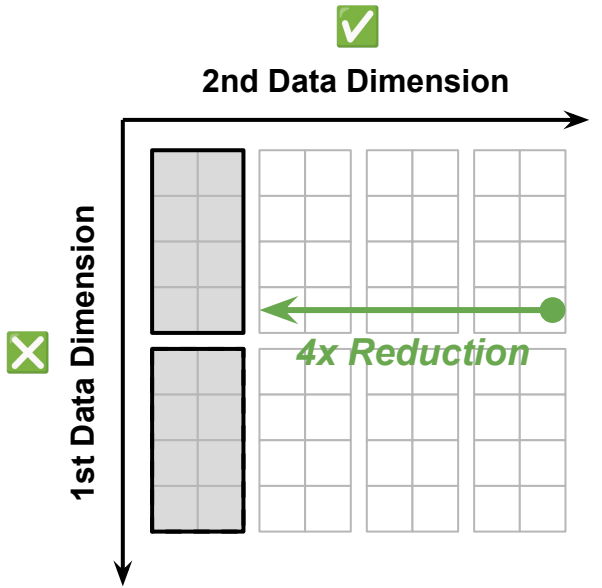
Element Shape ✅

Iteration Dimension ❌

(c) `itensor<4x2xf32,`

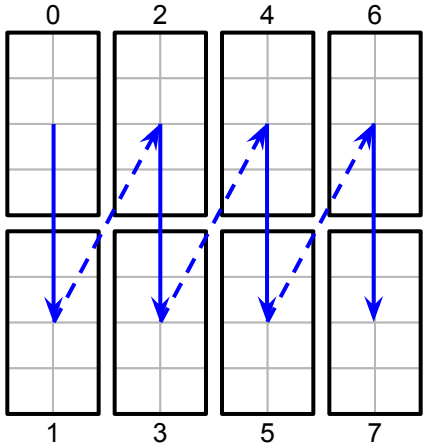
`iter_space: [4,2,2]*[2,1,4],`

`iter_map: (d0,d1,d2)->(d2,d0)>`



Traverse each data dimension from little-endian

Quiz



(a) `itensor<4x4xf32, iter_space: [4,2]*[2,4], iter_map: (d0,d1)->(d1,d0)>`

(b) `itensor<4x2xf32, iter_space: [4,2]*[2,4], iter_map: (d0,d1)->(d1,d0)>`

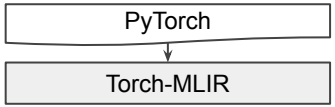
(c) `itensor<2x2xf32, iter_space: [4,2]*[2,4], iter_map: (d0,d1)->(d1,d0)>`

(d) `itensor<2x1xf32, iter_space: [4,2]*[2,4], iter_map: (d0,d1)->(d1,d0)>`

StreamTensor Outline

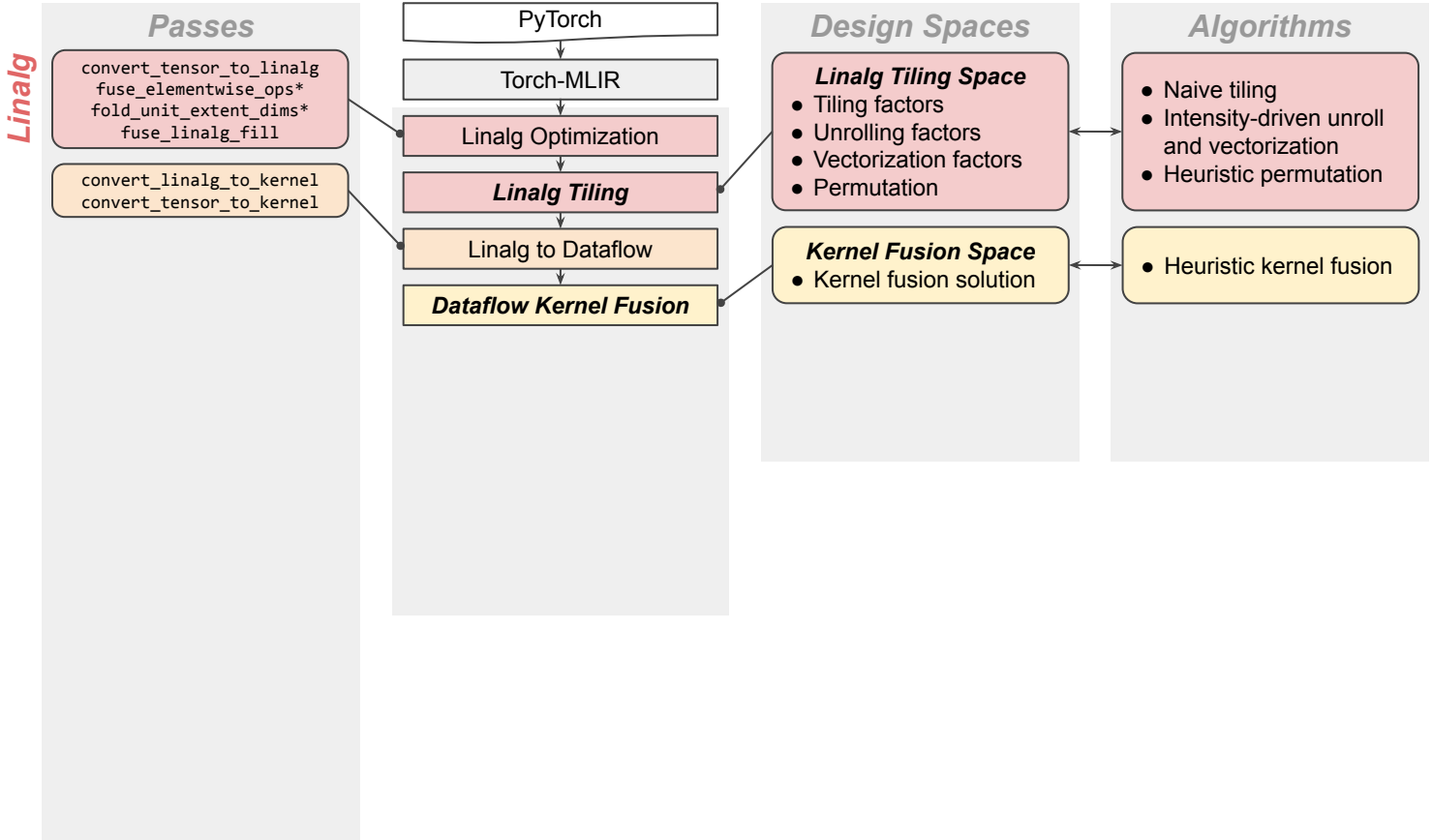
- Motivation
- StreamTensor Typing System
- StreamTensor Compilation Pipeline
- StreamTensor Design Spaces
- StreamTensor Results

StreamTensor Framework Overview

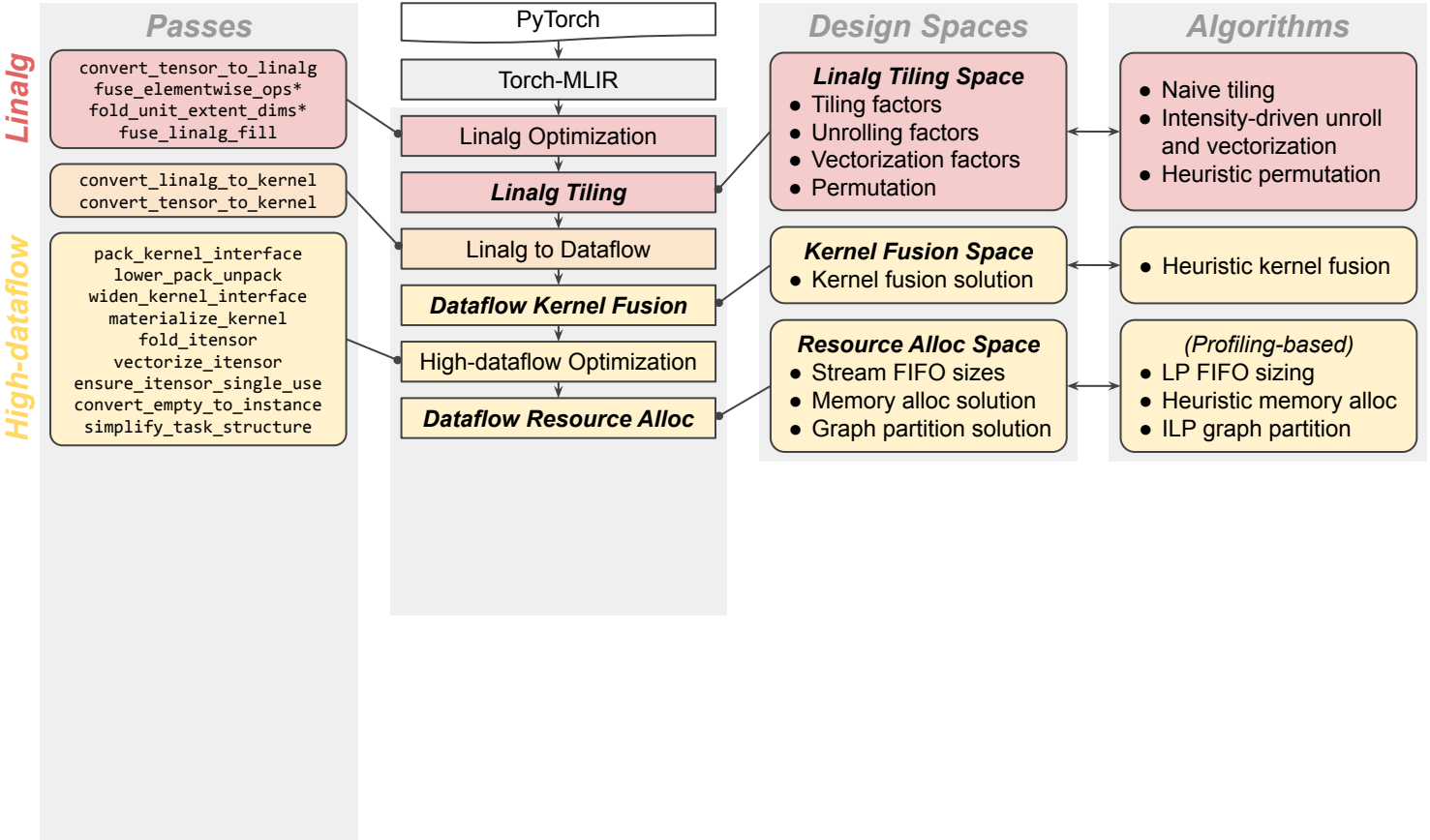


StreamTensor Framework Overview

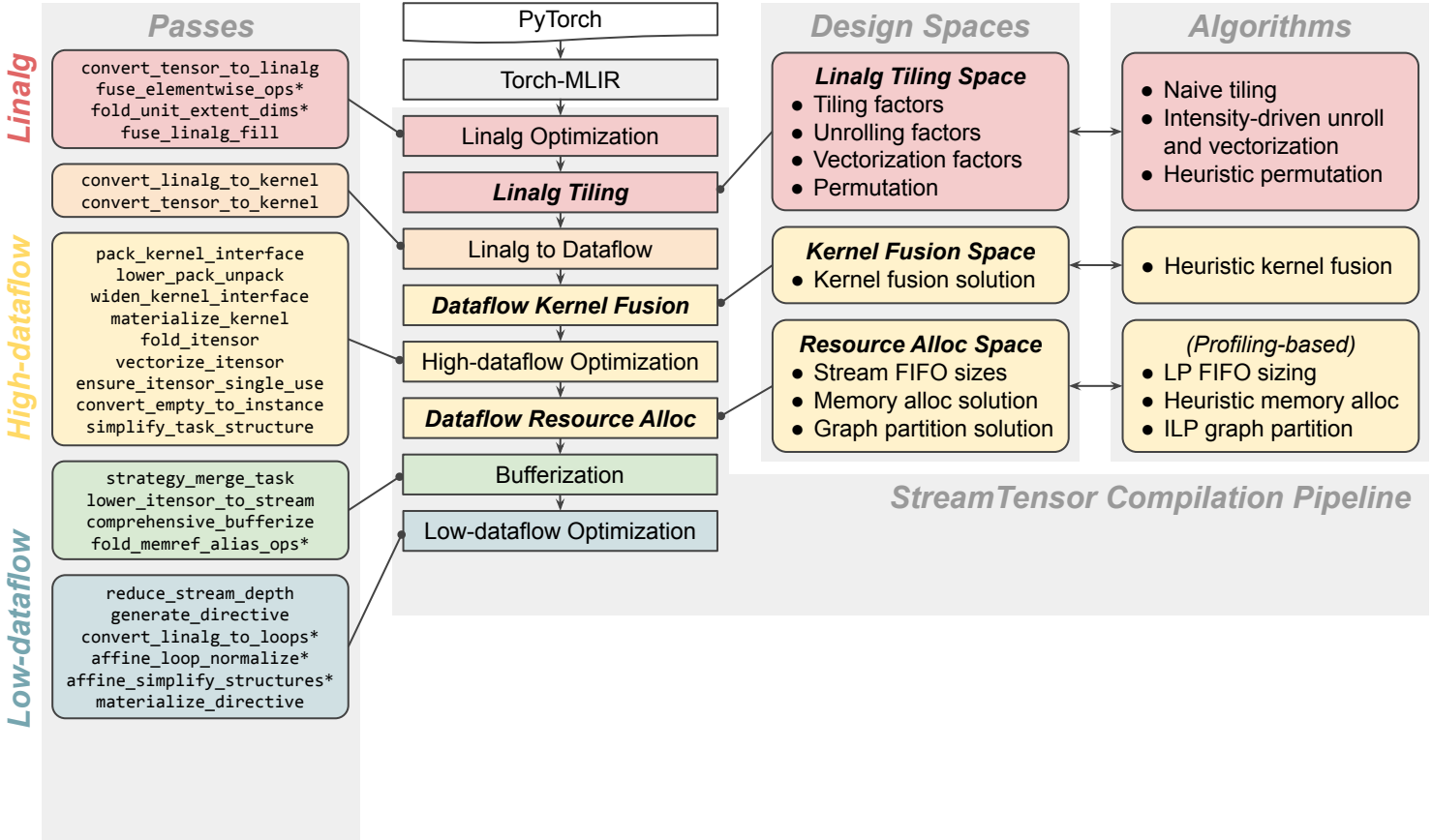
StreamTensor Framework Overview



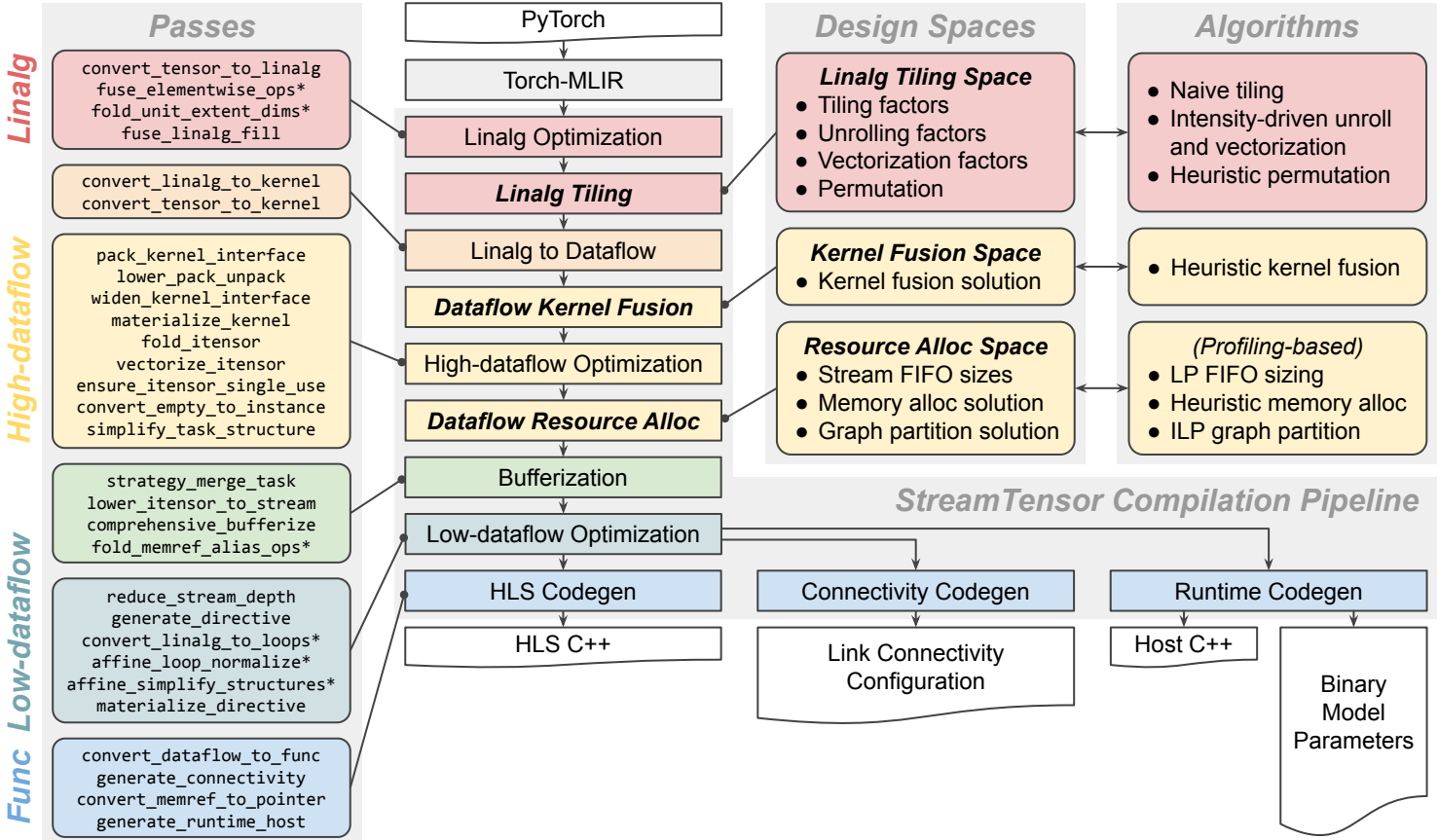
StreamTensor Framework Overview



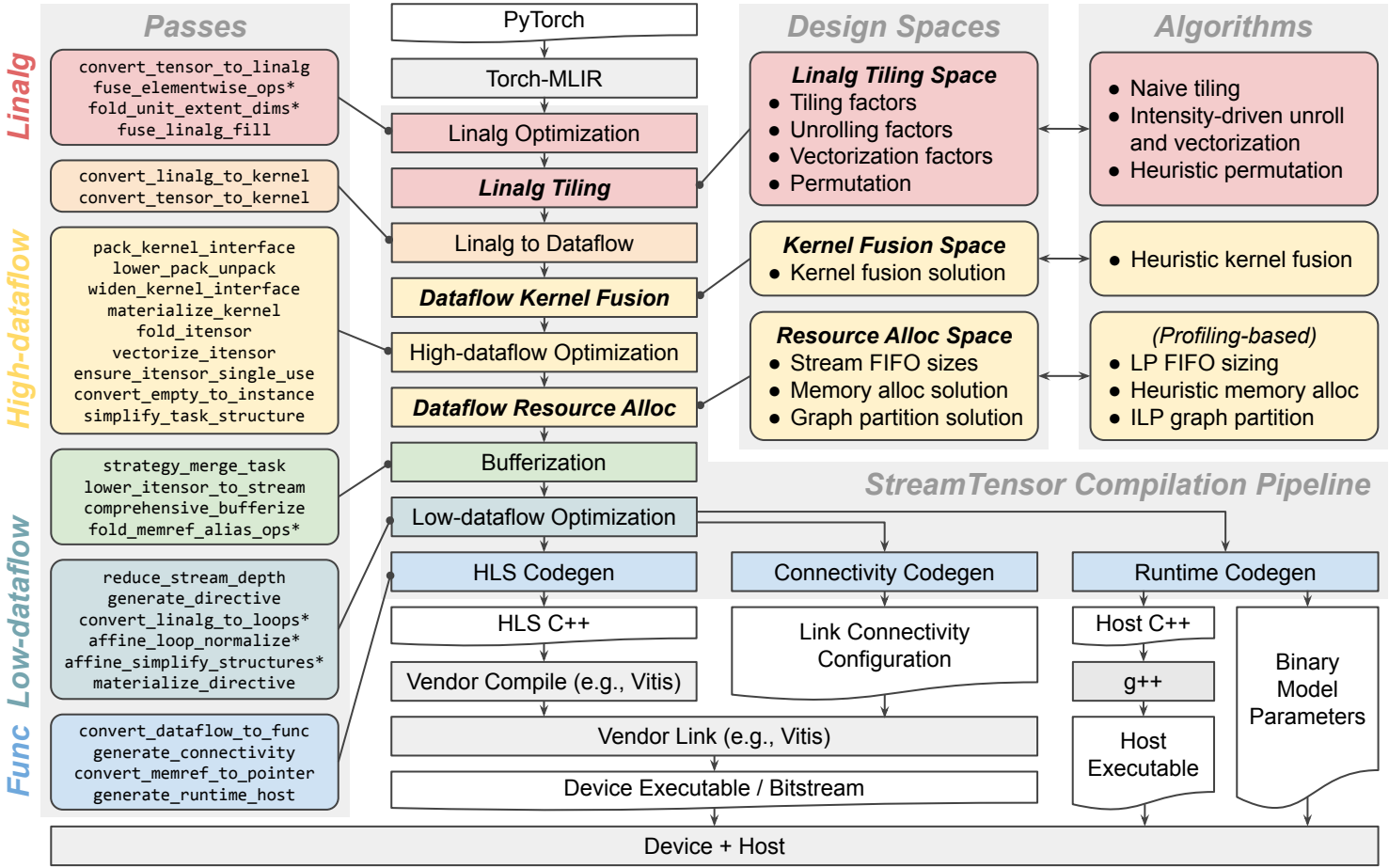
StreamTensor Framework Overview



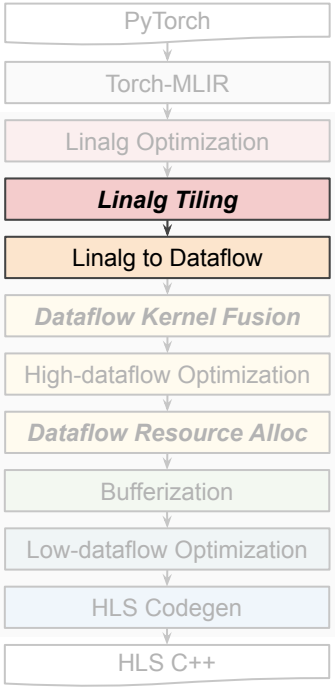
StreamTensor Framework Overview

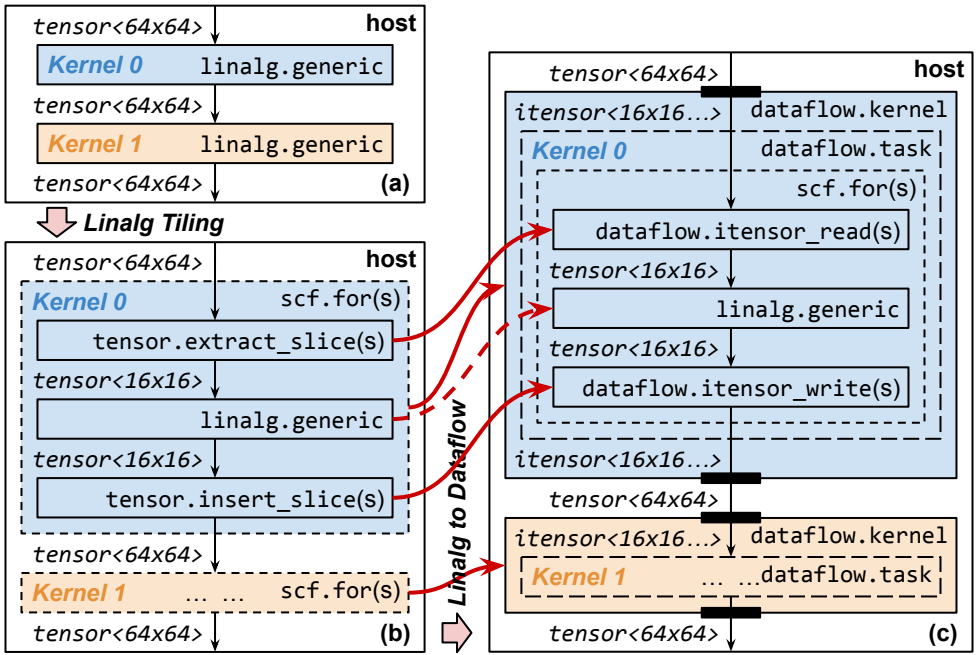


StreamTensor Framework Overview



Linalg Tiling + Linalg-to-Dataflow Conversion





`scf` = Structured Control Flow

`linalg` = Linear Algebra

Transformed

Unchanged

`dataflow.kernel`

- Isolated from above
- Convert tensor to/from itensor at the boundary, representing DMAs implicitly
- Contains a graph of tasks
- *Easy for kernel fusion*

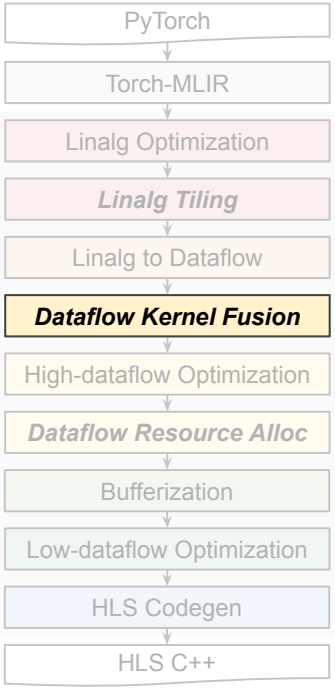
`dataflow.task`

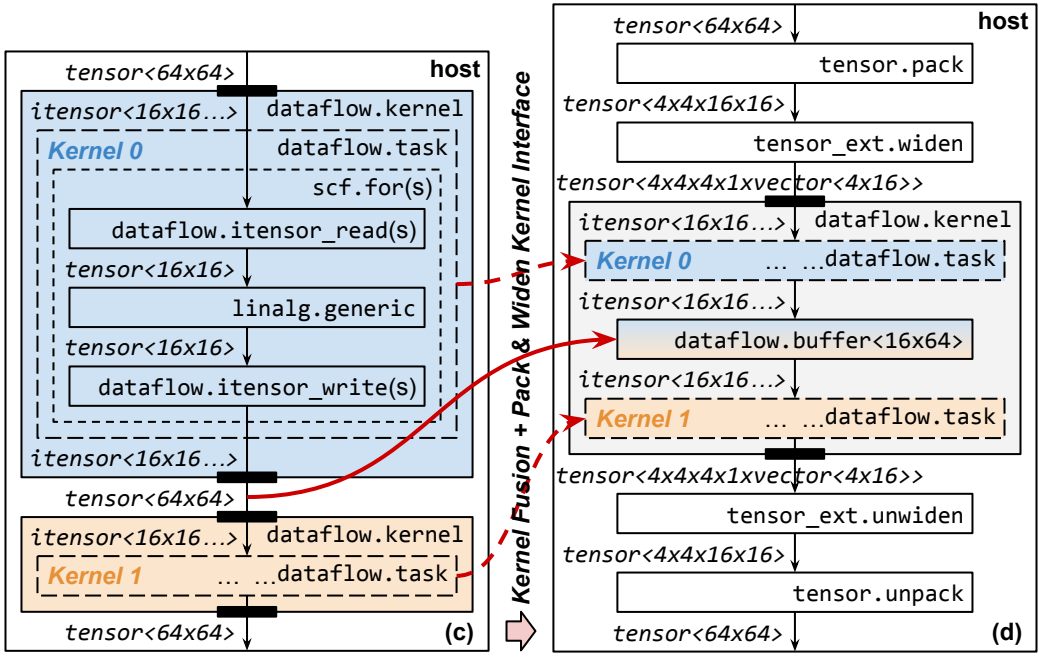
- Transparent from above
- Can be nested to represent a hierarchy of dataflow
- Contains a graph of operations
- Easy for task manipulation

`dataflow.itensor_read/write`

- Read a tensor slice/tile from an itensor (FIFO pull)
- Write a tensor slice/tile into an itensor (FIFO push)

Kernel Fusion + Pack & Widen Kernel Interface





`scf` = Structured Control Flow

`linalg` = Linear Algebra

Transformed

Unchanged

`dataflow.buffer`

- Convert an input itensor to another itensor with different layout

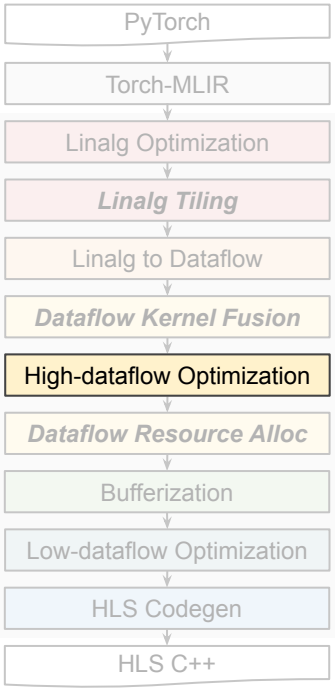
`tensor.pack/unpack`

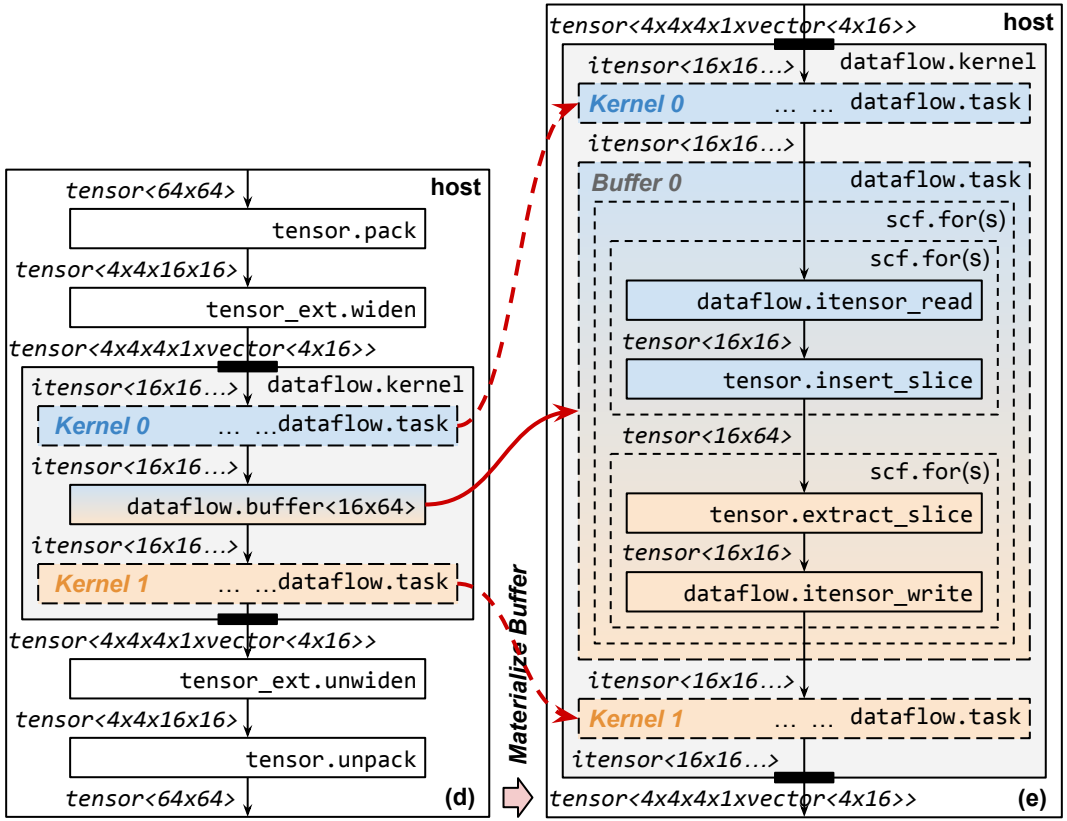
- Pack tensor from normal layout to tiled layout
- *Improve external memory access efficiency*

`tensor_ext.widen/unwiden`

- Widen tensor from scalar element to vector element
- *Improve external memory access bitwidth*

Buffer Materialization





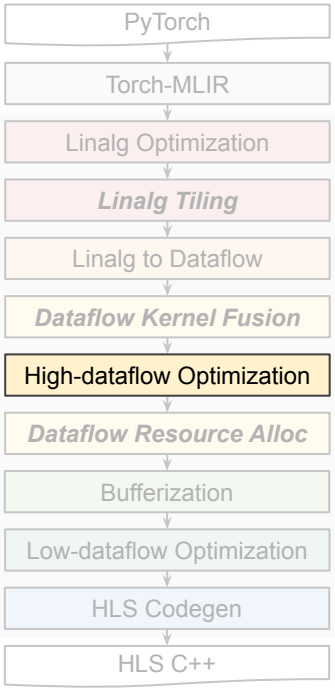
Before Materialization

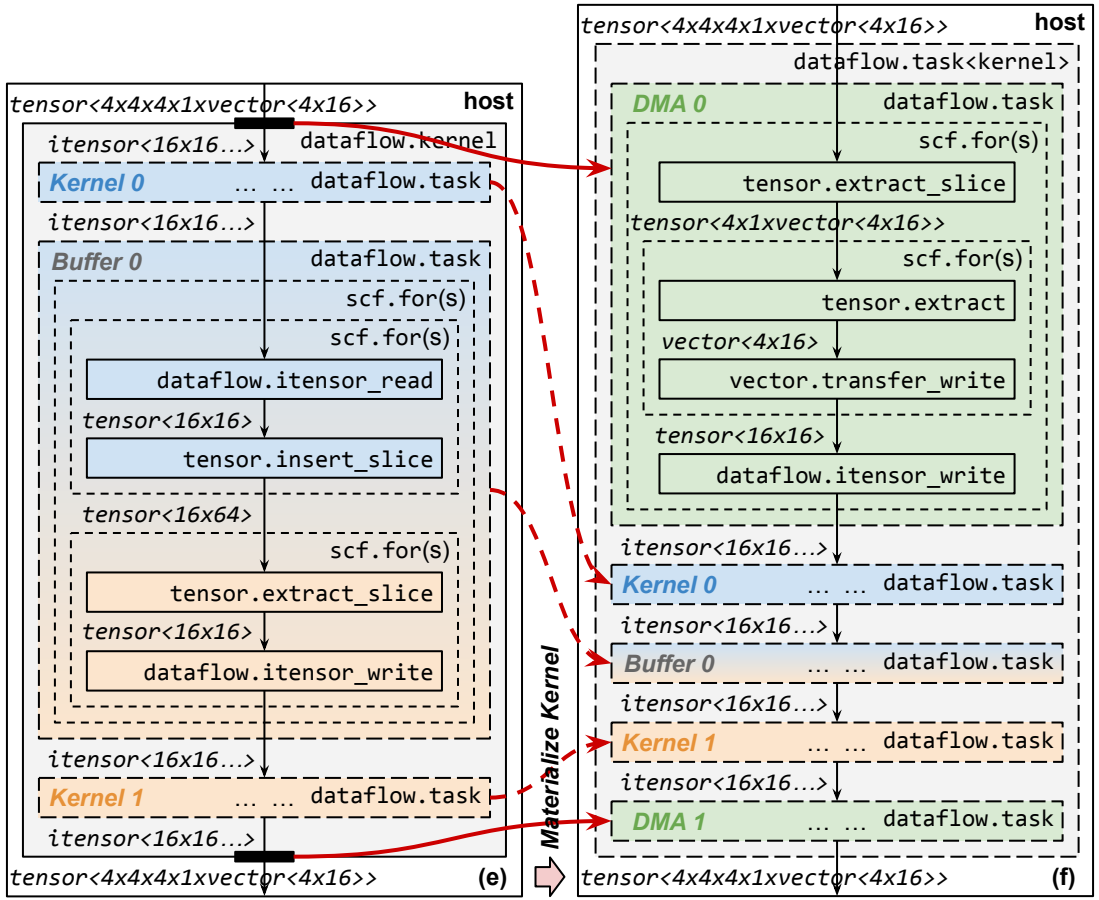
- Buffer is represented by a single operation
- *Easy for optimizations like CSE*

After Materialization

- Buffer is represented by task, loops, and tensor or itensor operations
- *Easy for subsequent dataflow optimizations*

Kernel Materialization





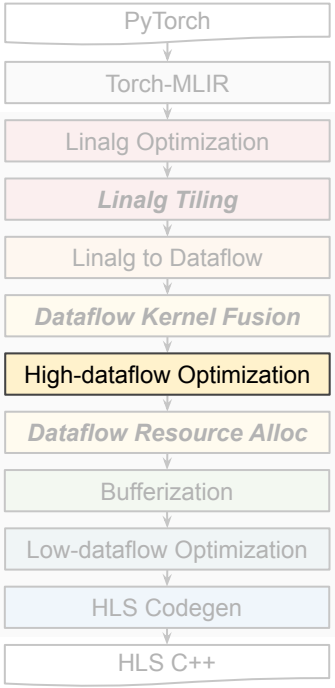
Before Materialization

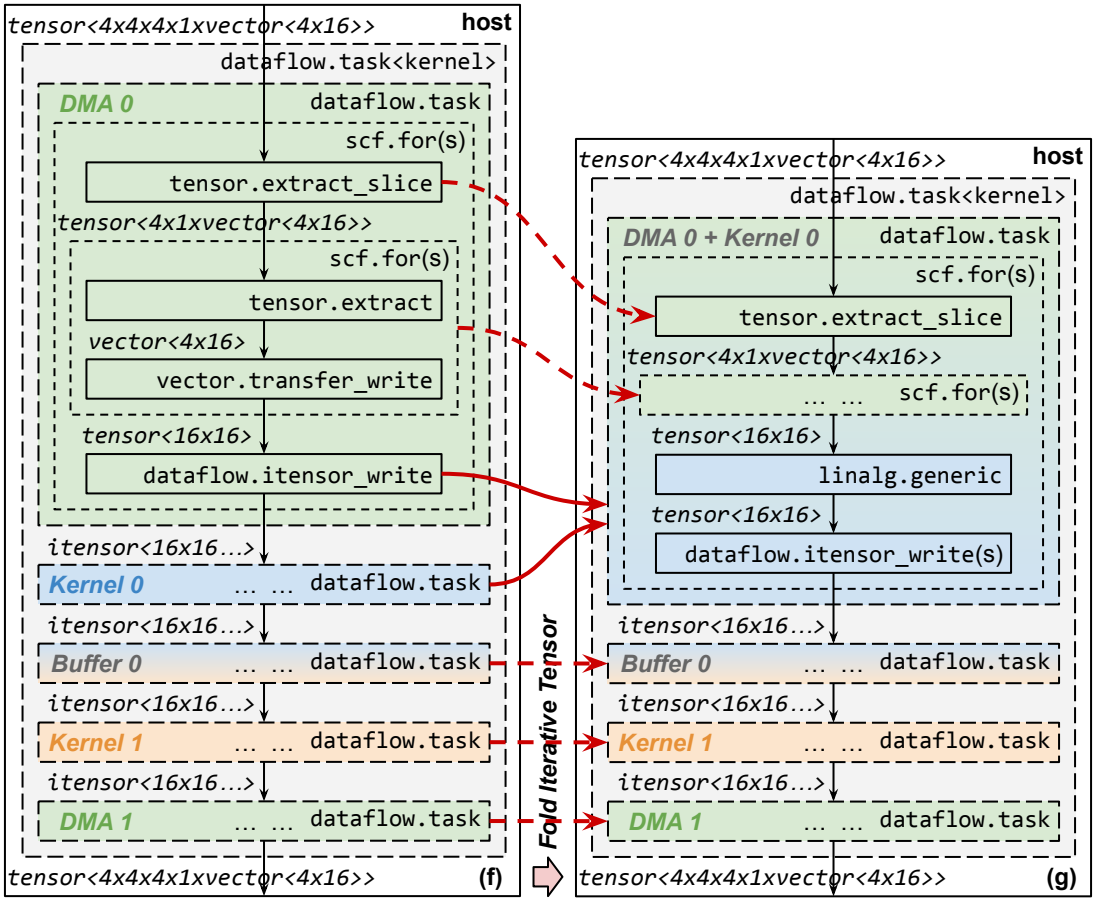
- DMA is represented implicitly at boundary of kernels
- *Easy for kernel fusion*

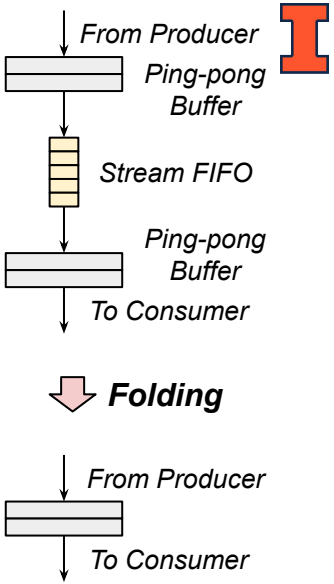
After Materialization

- DMA is represented by task, loops, and tensor or itensor operations
- *Easy for subsequent dataflow optimizations*

Fold Iterative Tensor



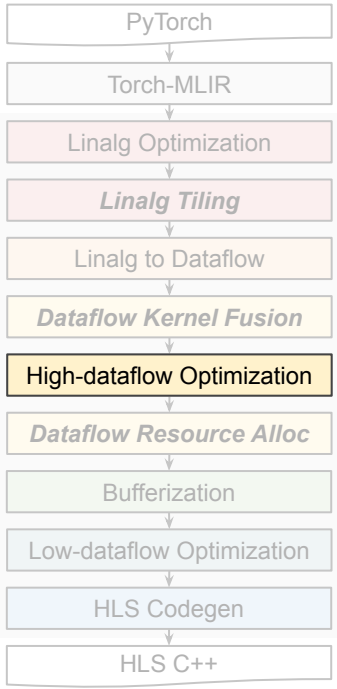


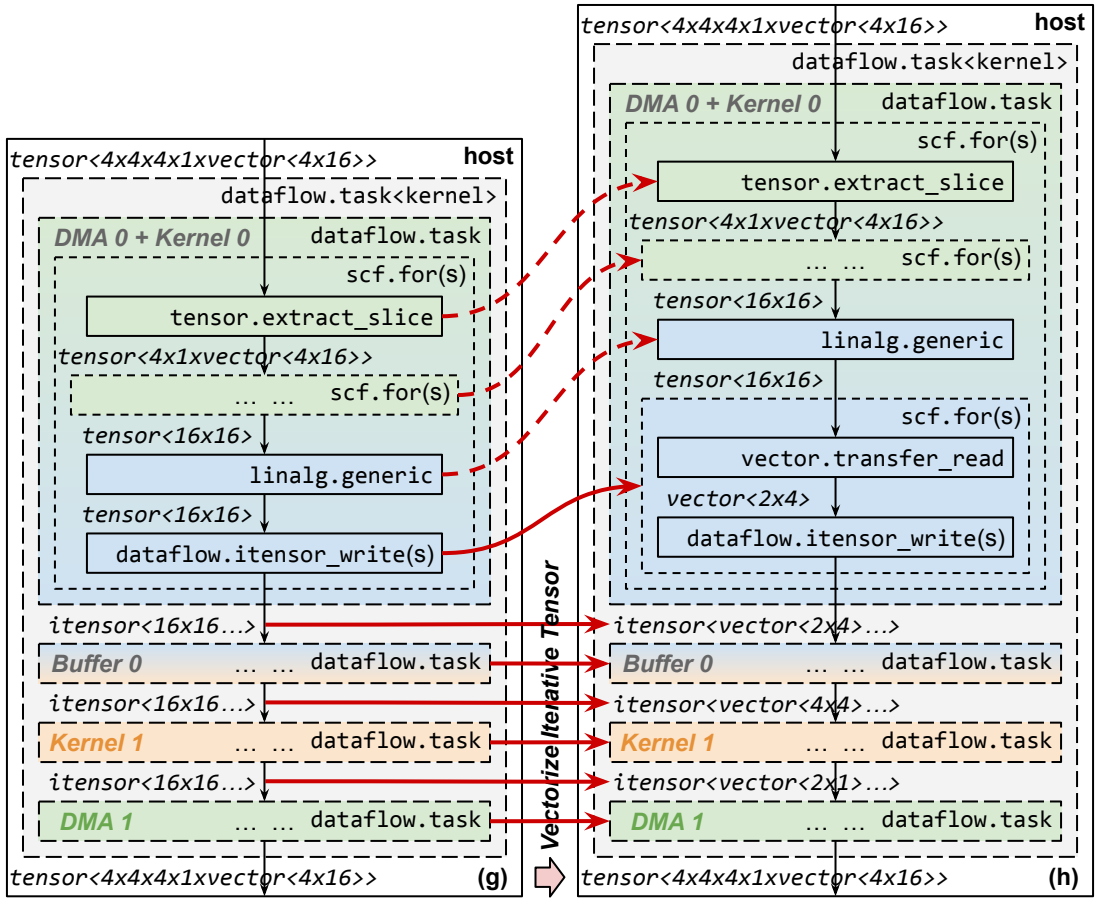


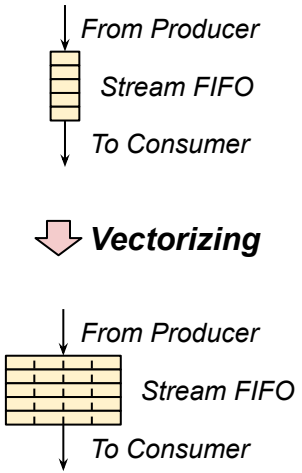
Benefits

- Merge two local ping-pong buffers into one, reducing on-chip memory
- Increase the overlap between the tasks, reducing latency

Vectorize Iterative Tensor







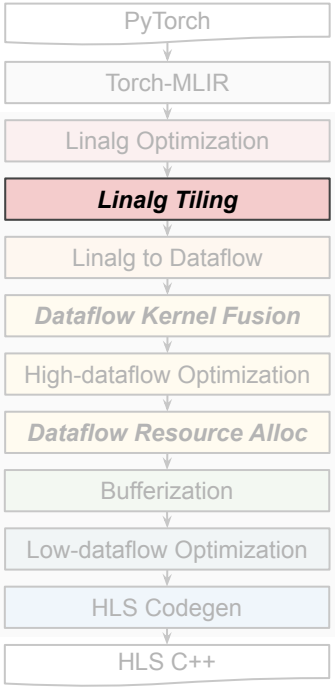
Benefits

- Increase the bitwidth of FIFOs
- Match computation parallelization
- Match external memory bandwidth

StreamTensor Outline

- Motivation
- StreamTensor Typing System
- StreamTensor Compilation Pipeline
- StreamTensor Design Spaces
- StreamTensor Results

Linalg Tiling Space




```
1  float A[32][16];
2  NODE0_I: for (int i=0; i<32; i++)
3    NODE0_K: for (int k=0; k<16; k++)
4      A[i][k] = ...; // Load array A.
5
6  float B[16][16];
7  NODE1_K: for (int k=0; k<16; k++)
8    NODE1_J: for (int j=0; j<16; j++)
9      B[k][j] = ...; // Load array B.
10
11 float C[16][16];
12 NODE2_I: for (int i=0; i<16; i++)
13   NODE2_J: for (int j=0; j<16; j++)
14     NODE2_K: for (int k=0; k<16; k++)
15       C[i][j] = A[i*2][k] * B[k][j];
```

HIDA IA+CA Algorithm

Node	Intensity	Parallel Factor		Loop Unroll Factors			
		w/o IA	w/ IA	IA+CA	IA	CA	Naive
Node0	512	32	4	[4, 1]	[2, 2]	[8, 4]	[4, 8]
Node1	256	32	2	[1, 2]	[1, 2]	[4, 8]	[4, 8]
Node2	4,096	32	32	[4, 8, 1]	[4, 8, 1]	[4, 8, 1]	[4, 8, 1]

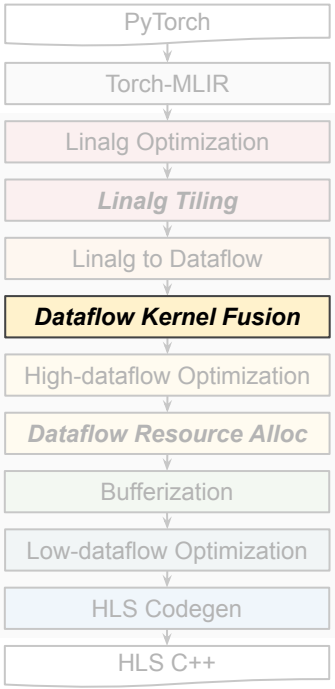
Intensity-aware (IA)
Connectedness-aware (CA)
HIDA DSE

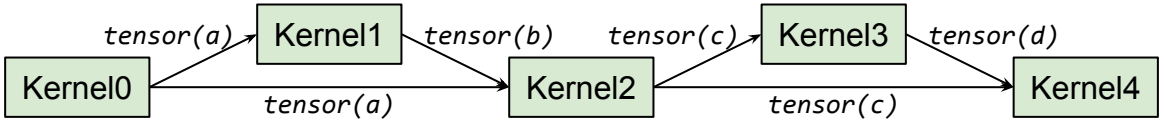
Naive
ScaleHLS
DSE

Array	Array Partition Factors				Bank Number				
	IA+CA	IA	CA	Naive	IA+CA	IA	CA	Naive	
A	[8, 1]	[8, 2]	[8, 4]	[8, 8]	8	16	32	64	*8x*
B	[1, 8]	[2, 8]	[4, 8]	[8, 8]	8	16	32	64	*8x*
C	[4, 8]	[4, 8]	[4, 8]	[4, 8]	32	32	32	32	*1x*

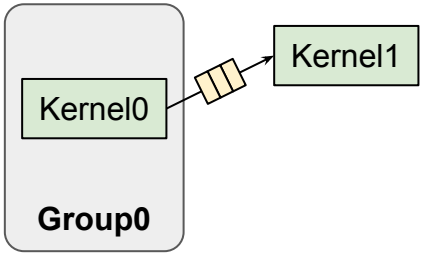
The design space is exposed at Python level, open for auto-tuning or other optimization algorithm

Kernel Fusion Space





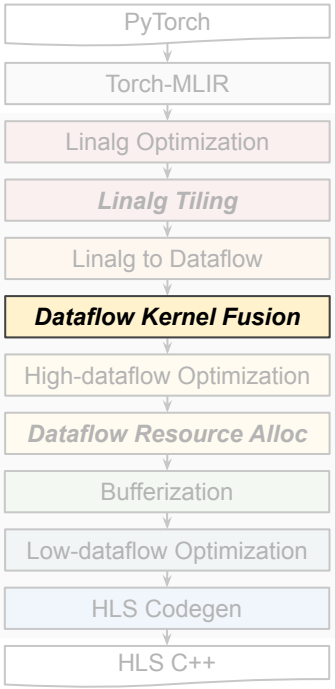
Topological Order Traversal

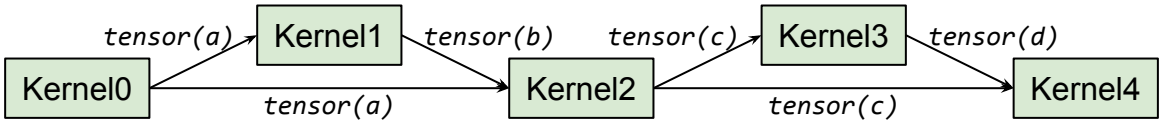


Group0 On-chip Memory Available

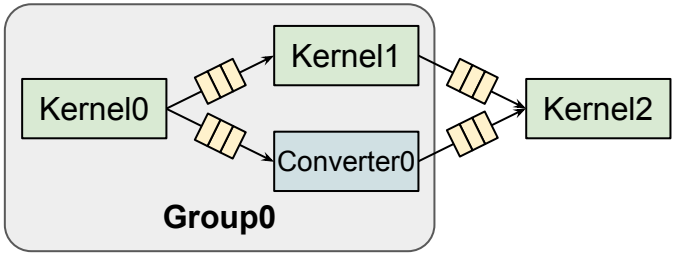
	Group0	Group1
Kernels	Kernel0	
Cost	$\text{Local}_{\text{Kernel0}}$	

Kernel Fusion Space (Cont.)





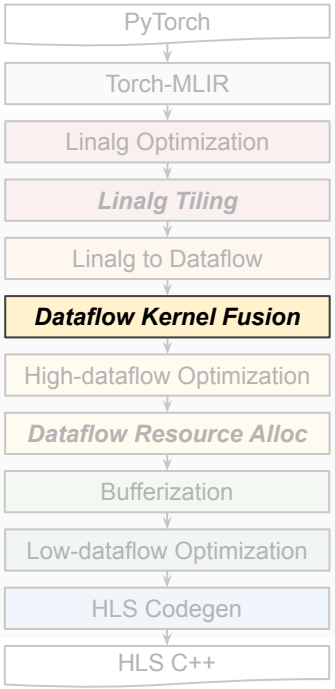
Topological Order Traversal

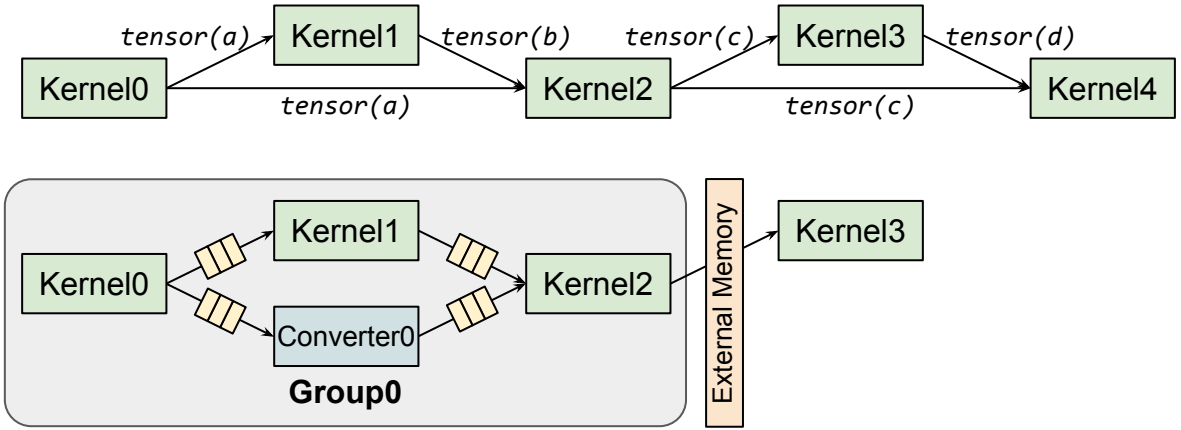


Group0 On-chip Memory Available

	Group0	Group1
Kernels	Kernel0 Kernel1	
Cost	$\text{Local}_{\text{Kernel0}}$ $\text{Local}_{\text{Kernel1}}$	

Kernel Fusion Space (Cont.)



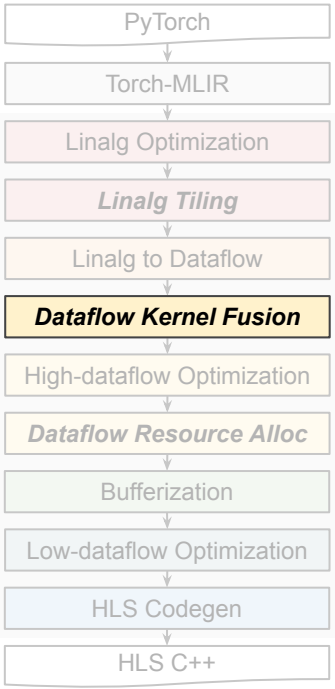


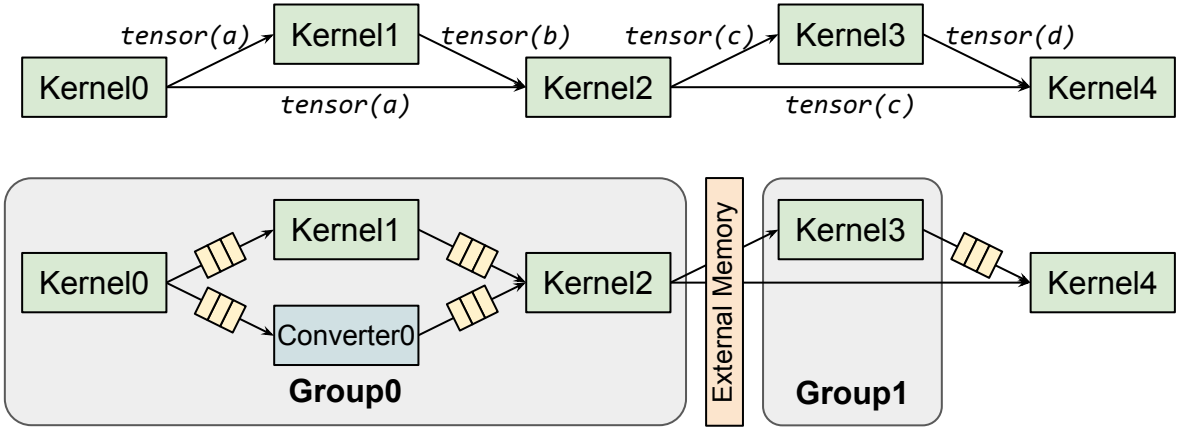
Topological Order Traversal

Group0 On-chip Memory Full

	Group0	Group1
Kernels	Kernel0 Kernel1 Kernel2	
Cost	$Local_{Kernel0}$ $Local_{Kernel1}$ Converter0 $Local_{Kernel2}$	

Kernel Fusion Space (Cont.)



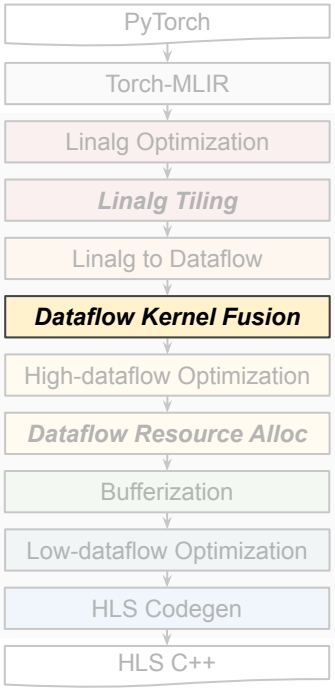


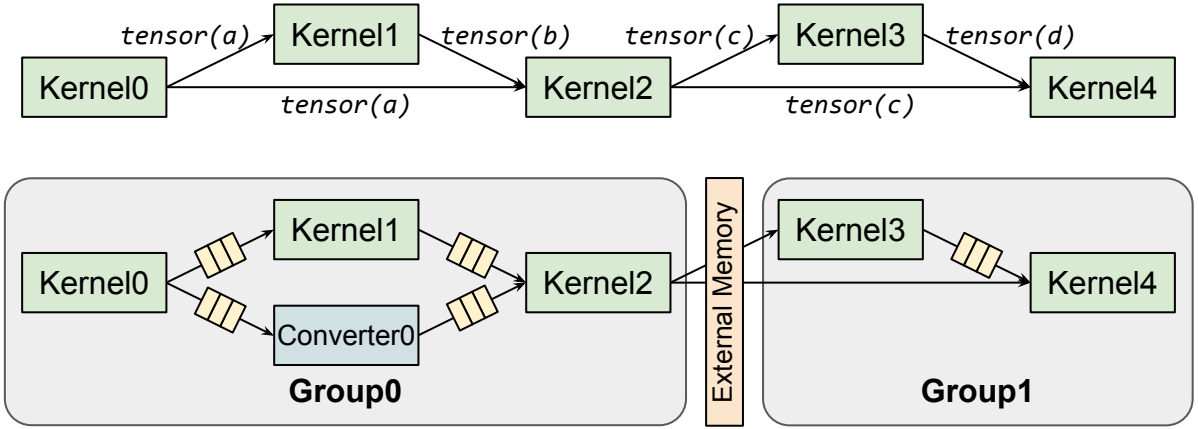
Topological Order Traversal

Select the Nearest Predecessor

	Group0	Group1
Kernels	Kernel0 Kernel1 Kernel2	Kernel3
Cost	$Local_{Kernel0}$ $Local_{Kernel1}$ Converter0 $Local_{Kernel2}$	$Local_{Kernel3}$

Kernel Fusion Space (Cont.)





Topological Order Traversal

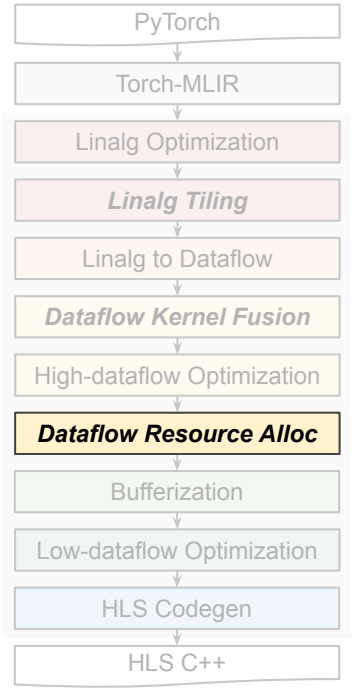
Intra-group: Streaming

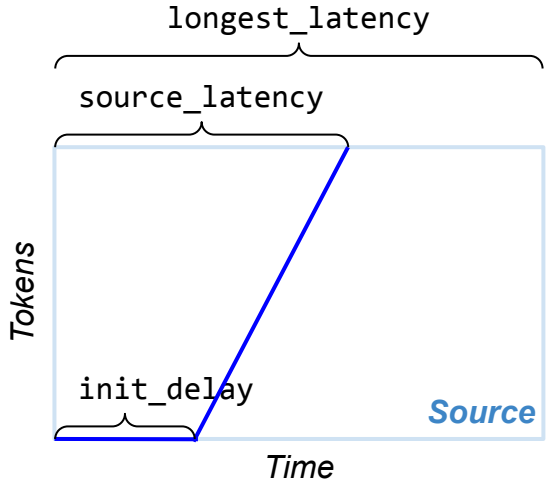
Inter-group: External Memory

	Group0	Group1
Kernels	Kernel0 Kernel1 Kernel2	Kernel3 Kernel4
Cost	$Local_{Kernel0}$ $Local_{Kernel1}$ Converter0 $Local_{Kernel2}$	$Local_{Kernel3}$ $Local_{Kernel4}$

Each Group ⬇ ***One Kernel*** ⬇ ***One Bitstream (if on FPGA)***

Resource Allocation Space - FIFO Sizing

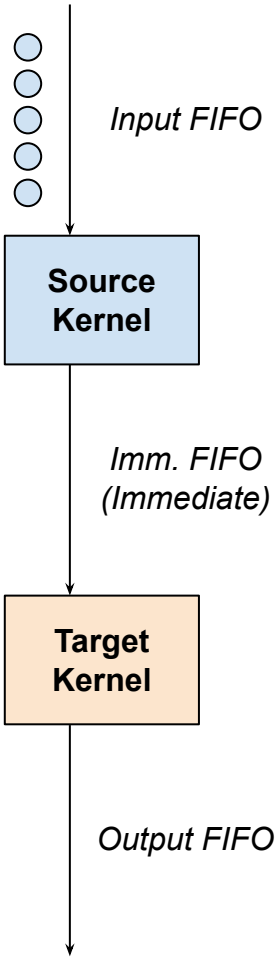




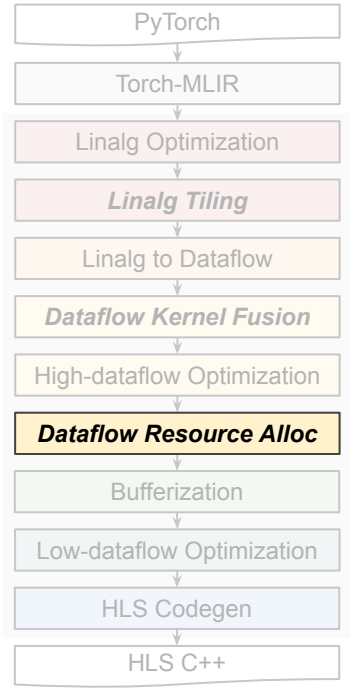
Source Token Production Curve (Push Imm. FIFO)

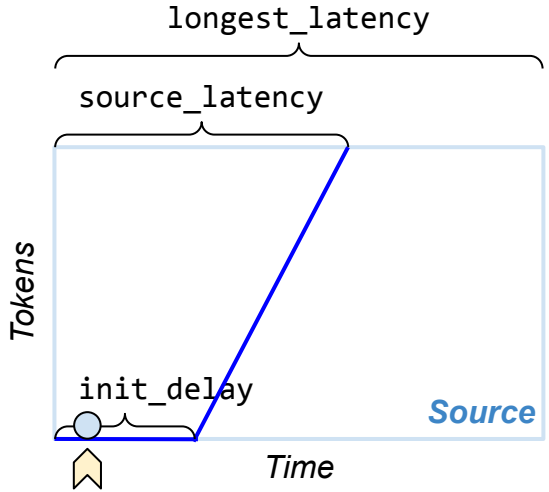
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling

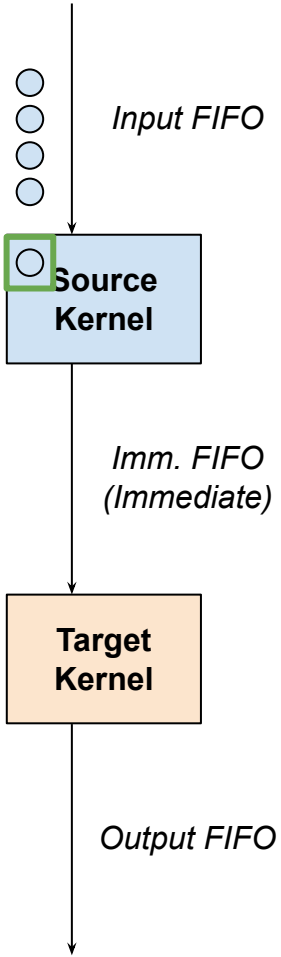




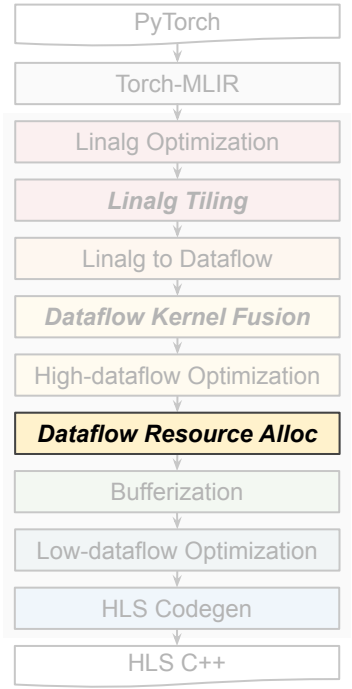
Source Token Production Curve (Push Imm. FIFO)

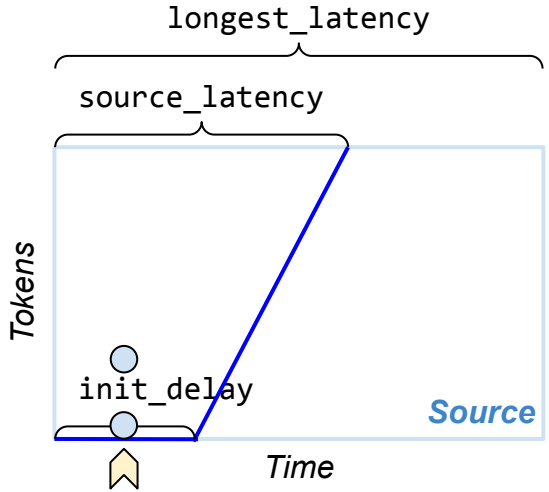
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

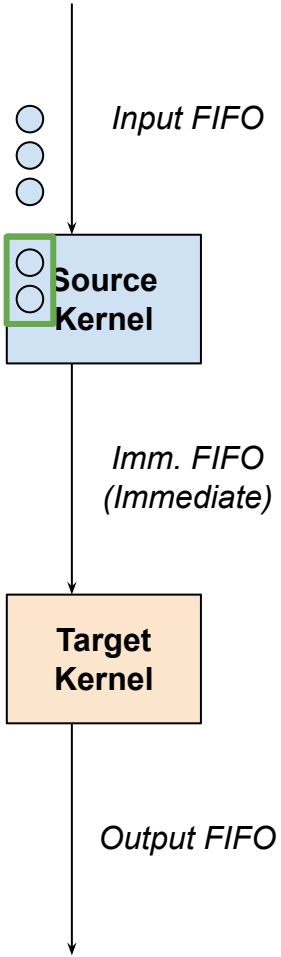




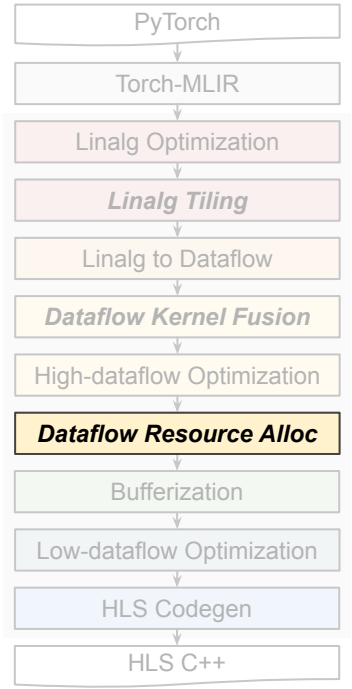
Source Token Production Curve (Push Imm. FIFO)

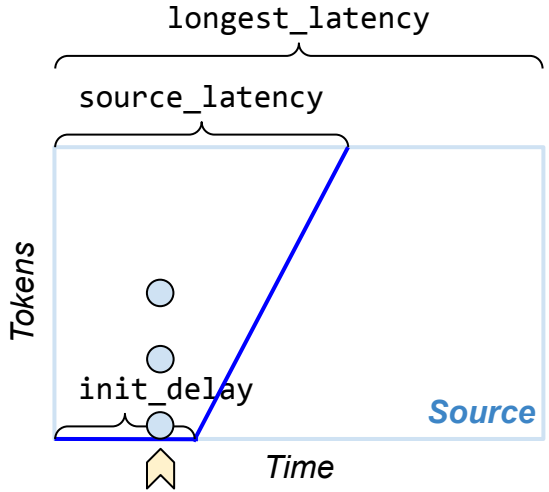
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

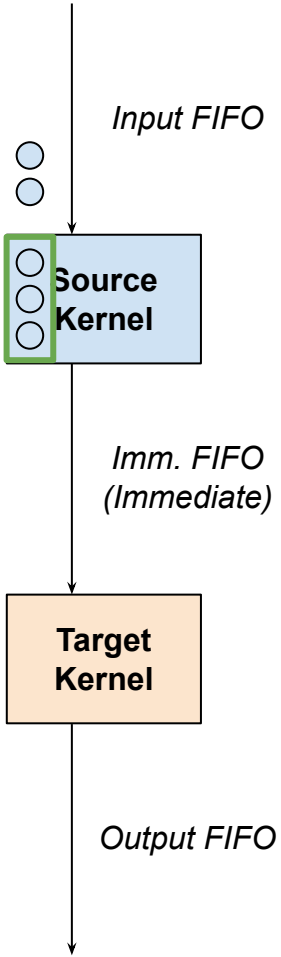




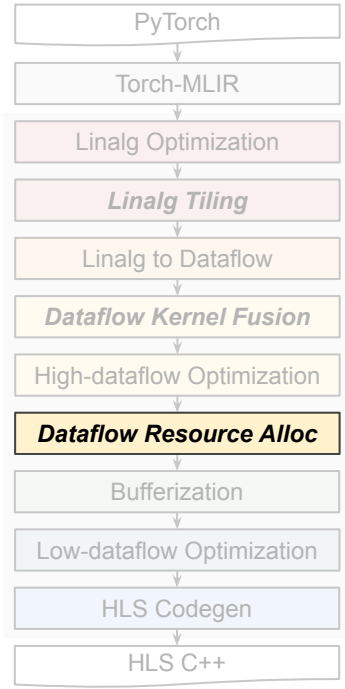
Source Token Production Curve (Push Imm. FIFO)

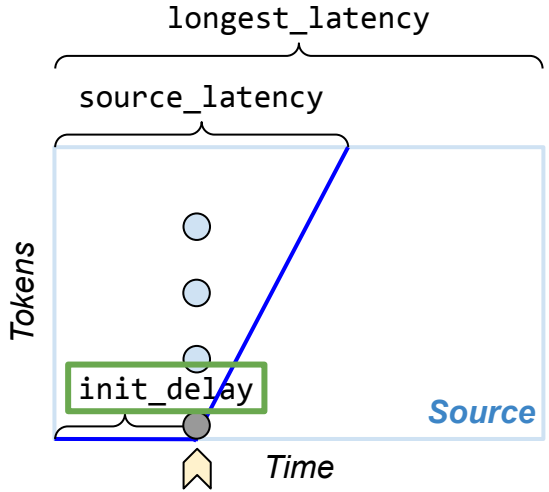
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

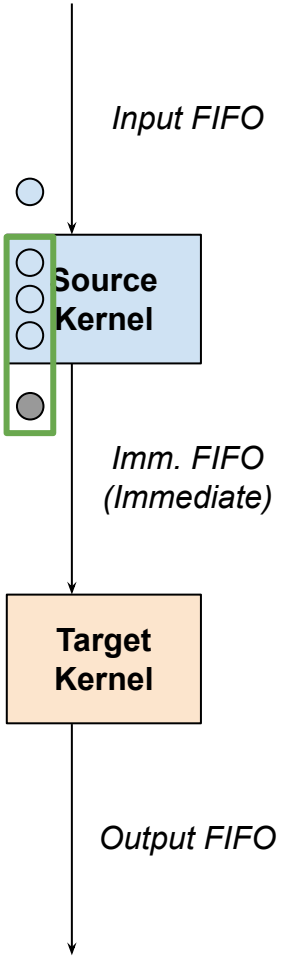




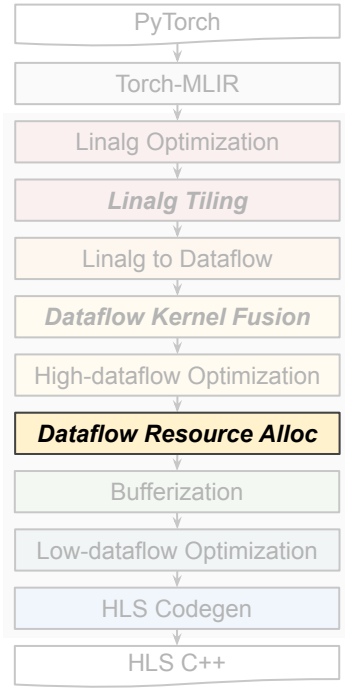
Source Token Production Curve (Push Imm. FIFO)

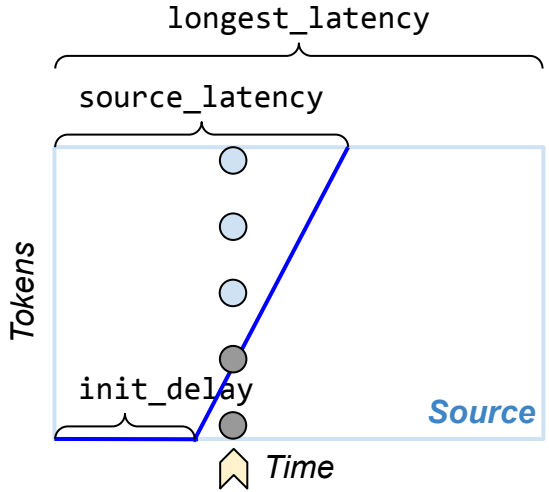
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

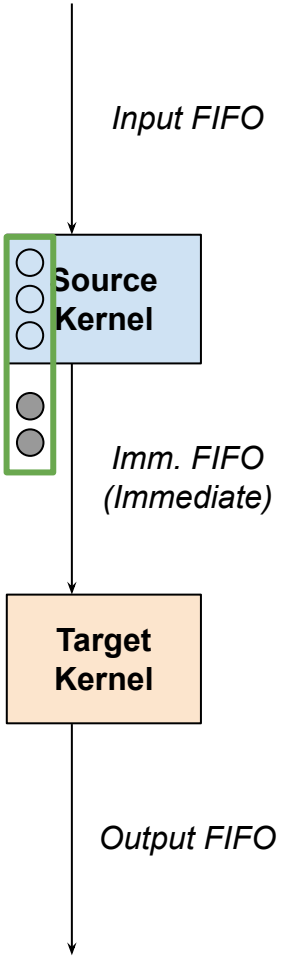




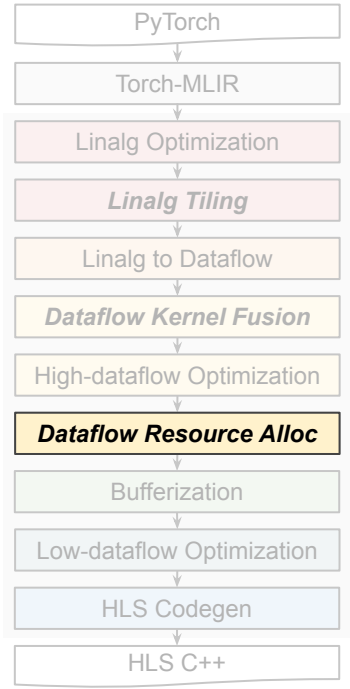
Source Token Production Curve (Push Imm. FIFO)

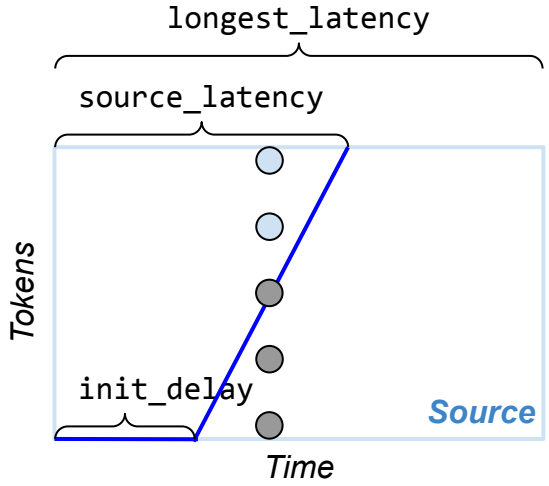
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

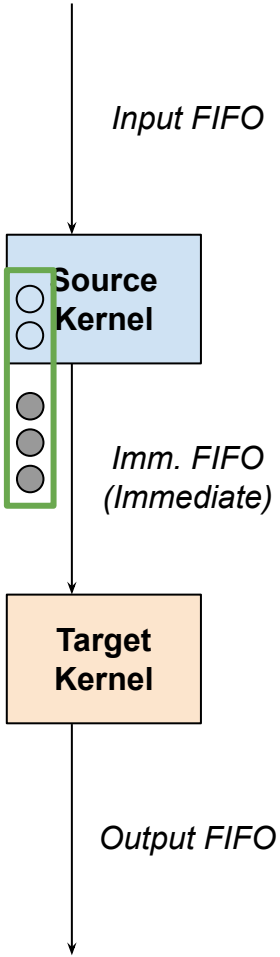




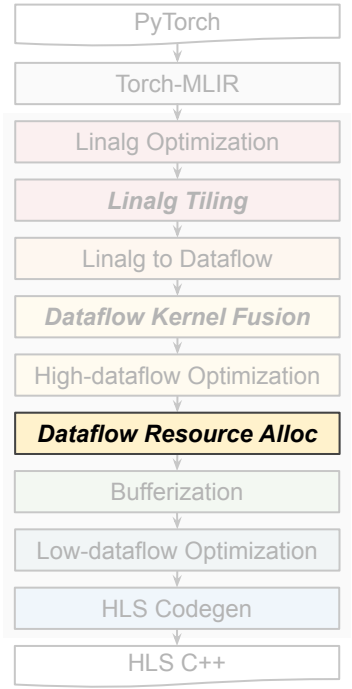
Source Token Production Curve (Push Imm. FIFO)

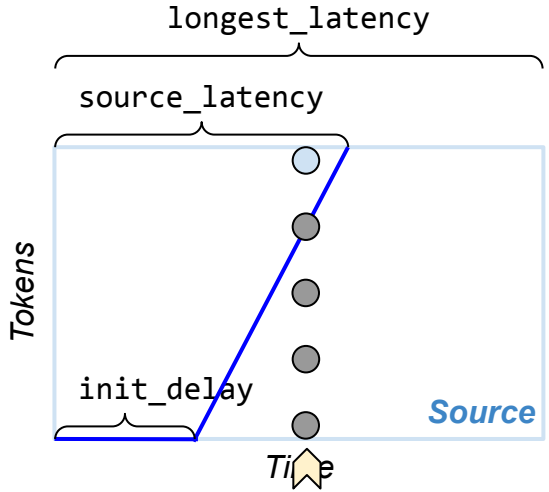
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

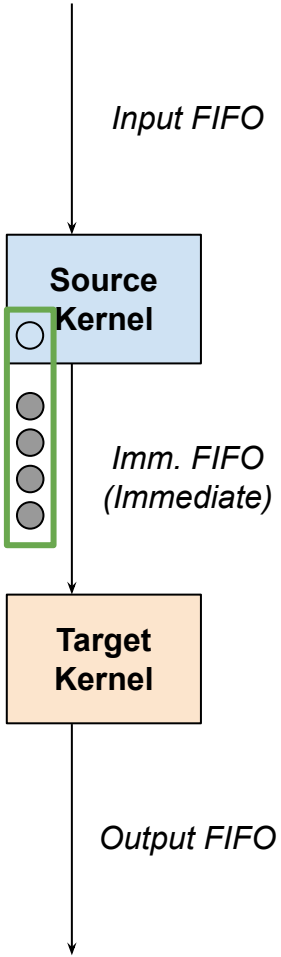




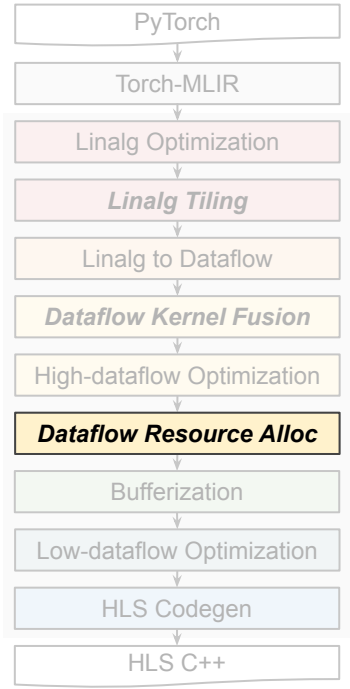
Source Token Production Curve (Push Imm. FIFO)

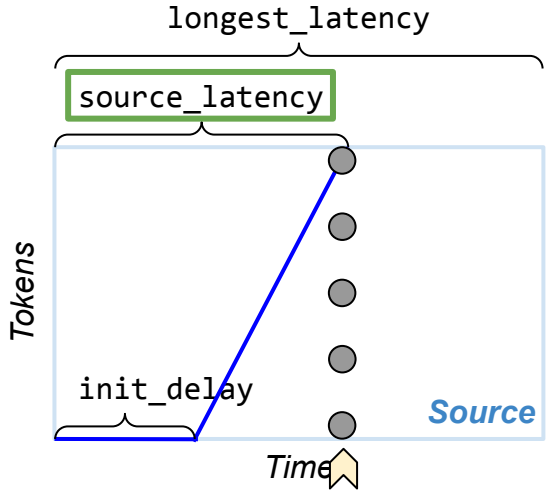
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

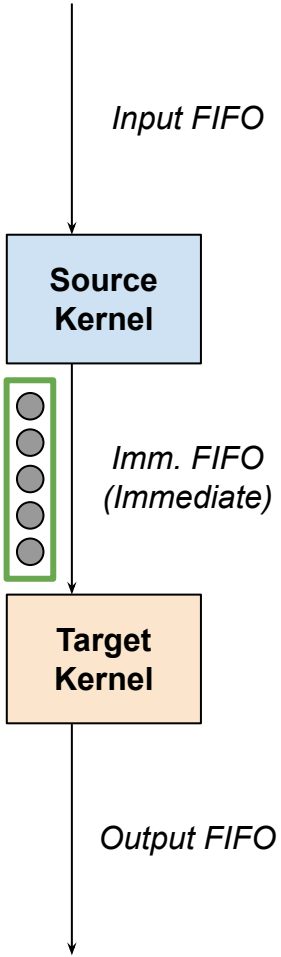




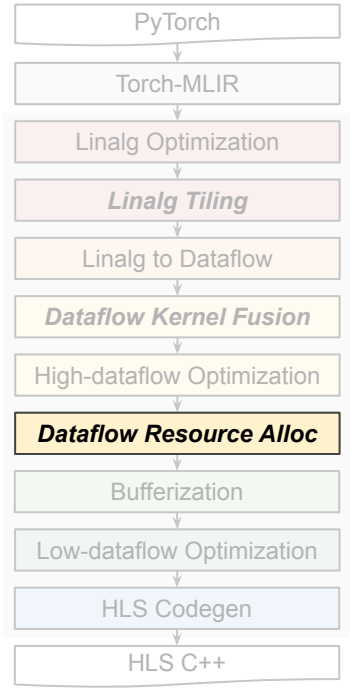
Source Token Production Curve (Push Imm. FIFO)

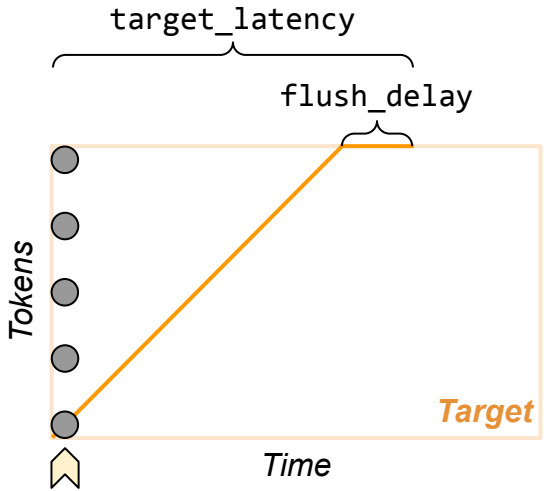
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

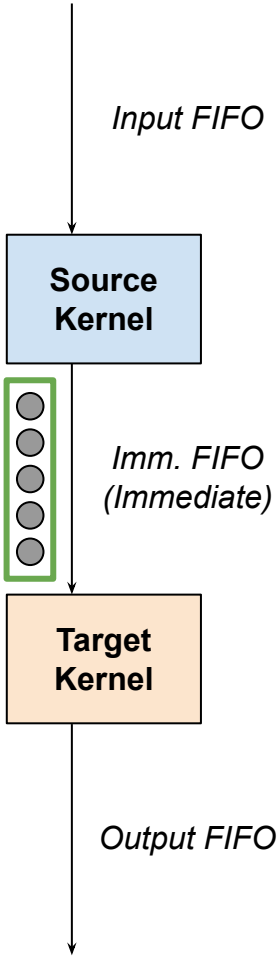




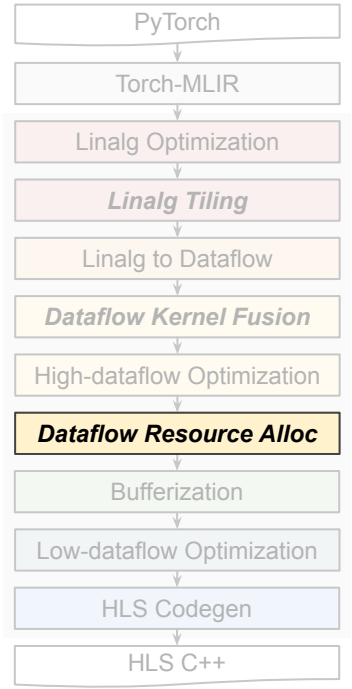
Target Token Consumption Curve (Pull Imm. FIFO)

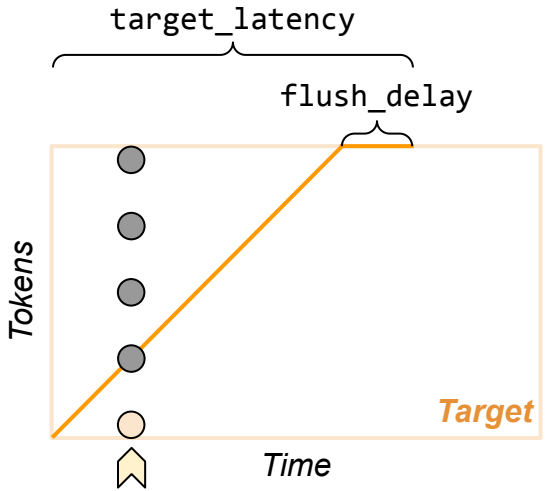
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

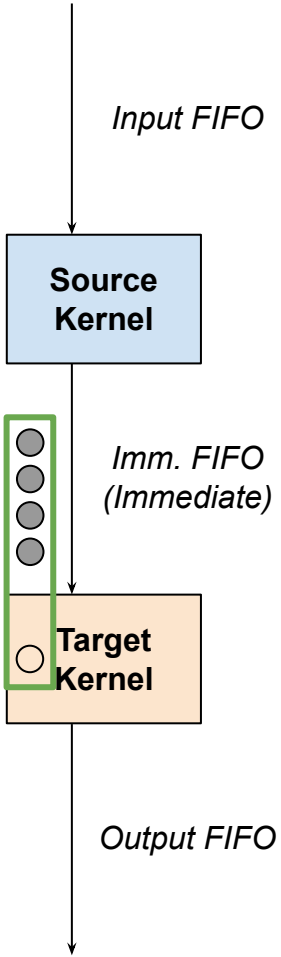




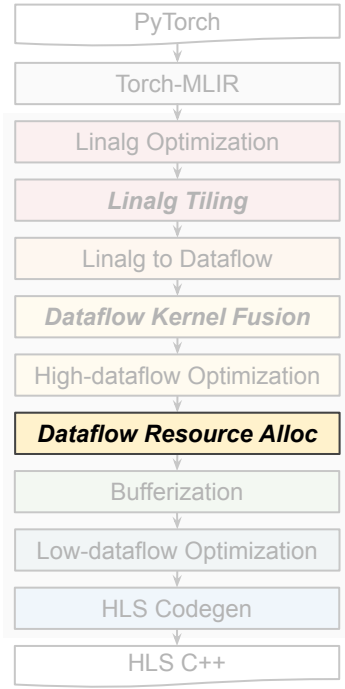
Target Token Consumption Curve (Pull Imm. FIFO)

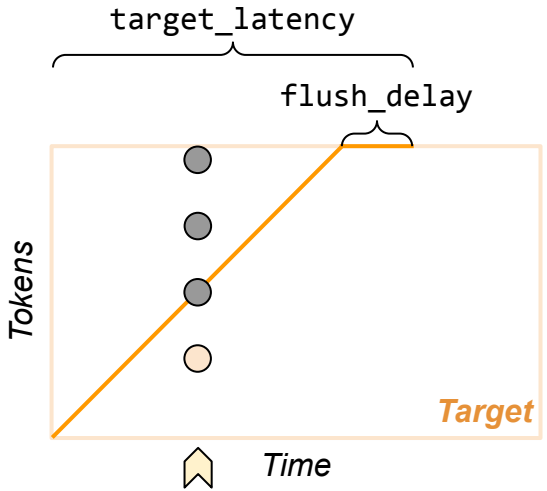
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

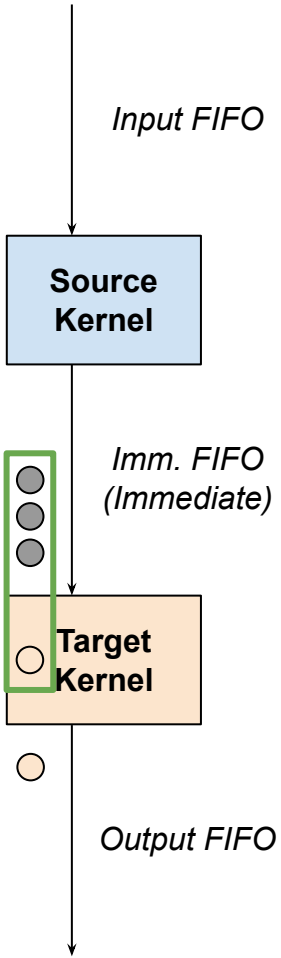




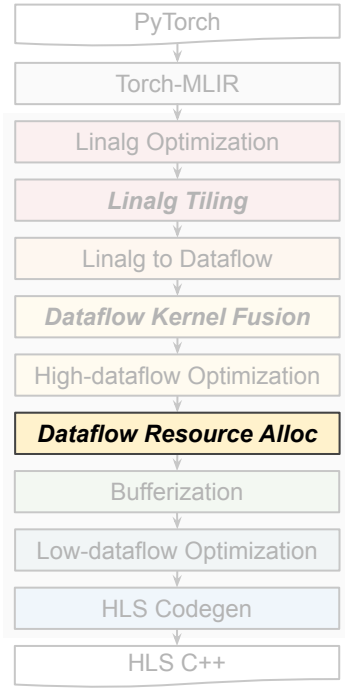
Target Token Consumption Curve (Pull Imm. FIFO)

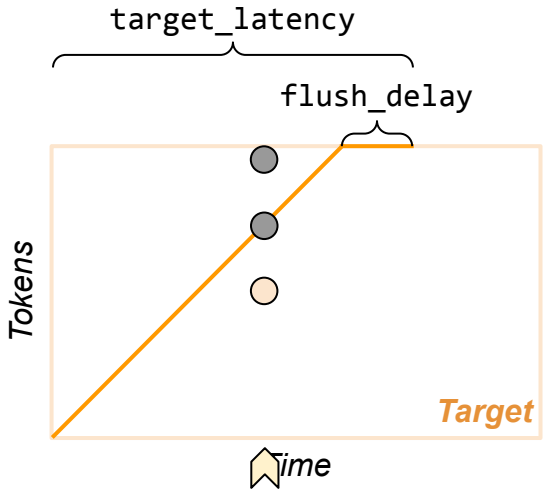
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

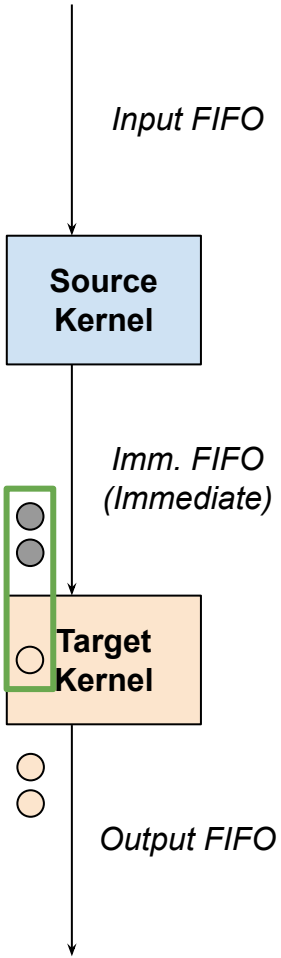




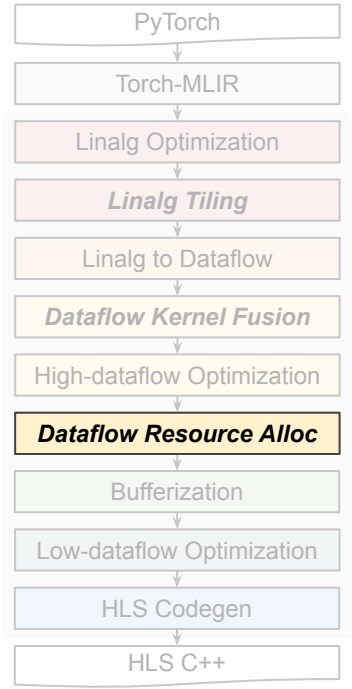
Target Token Consumption Curve (Pull Imm. FIFO)

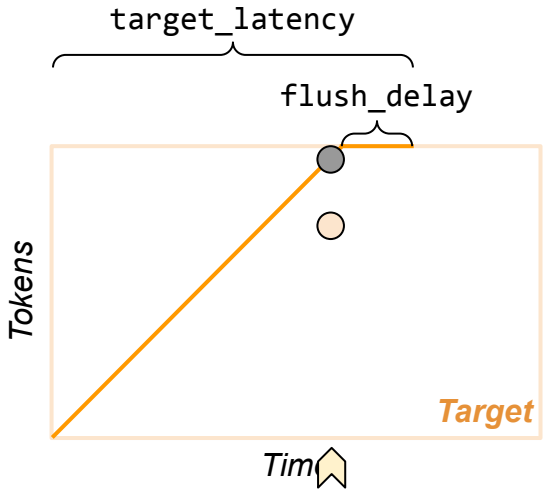
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

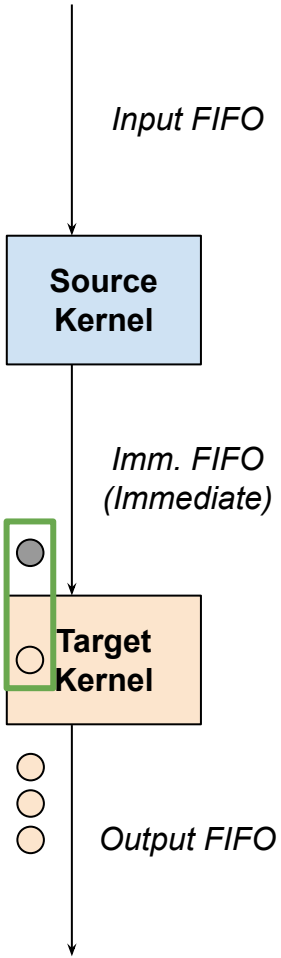




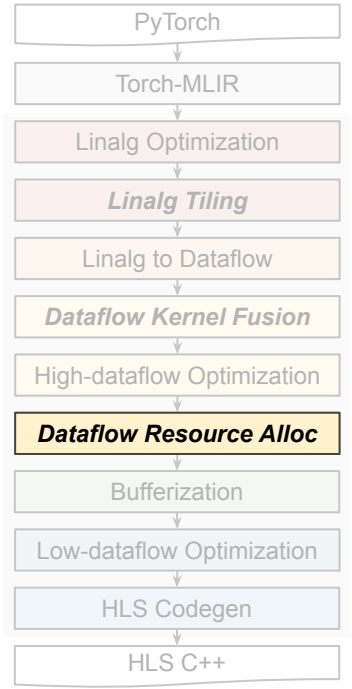
Target Token Consumption Curve (Pull Imm. FIFO)

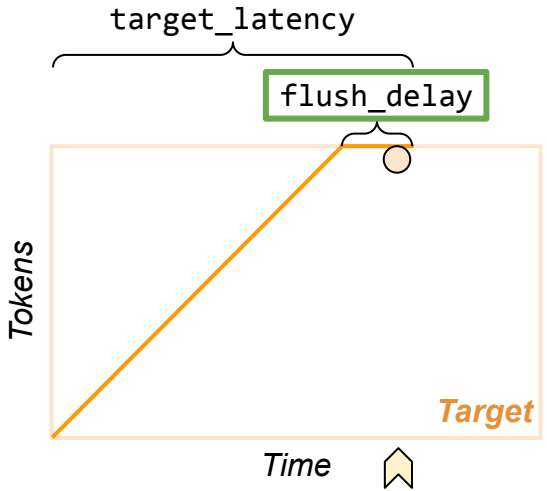
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

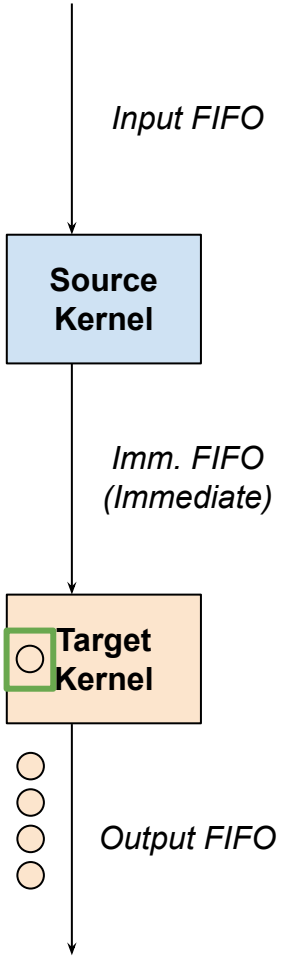




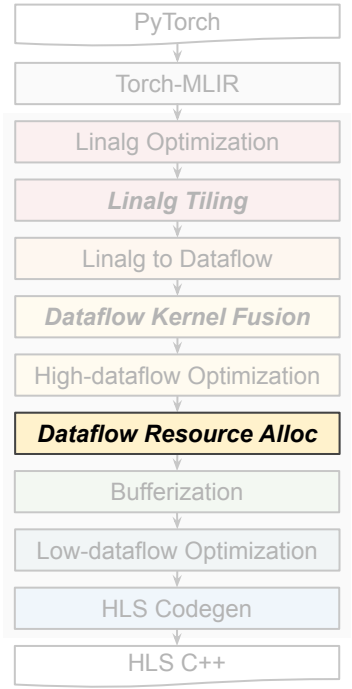
Target Token Consumption Curve (Pull Imm. FIFO)

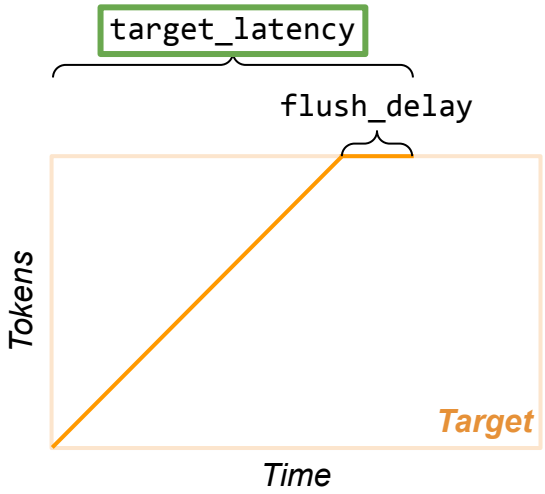
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

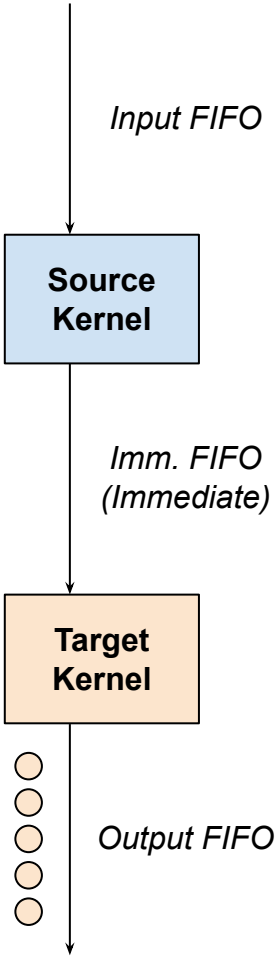




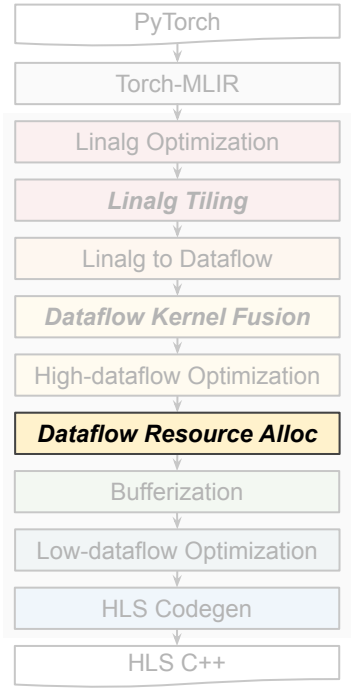
Target Token Consumption Curve (Pull Imm. FIFO)

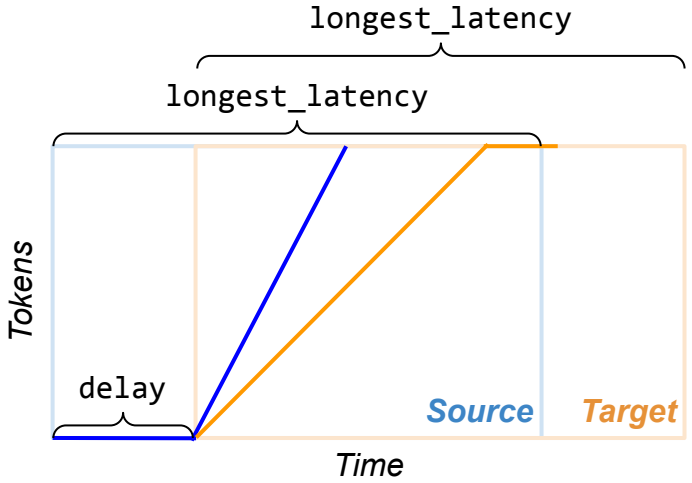
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

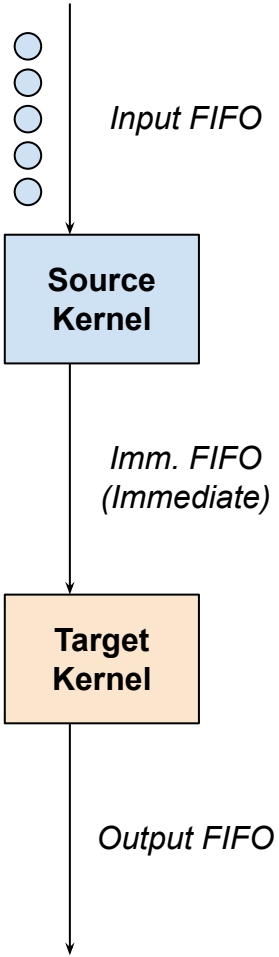




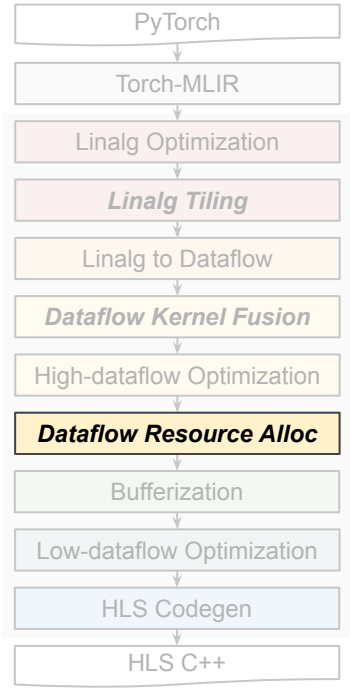
Overall Token Status Curve (Imm. FIFO)

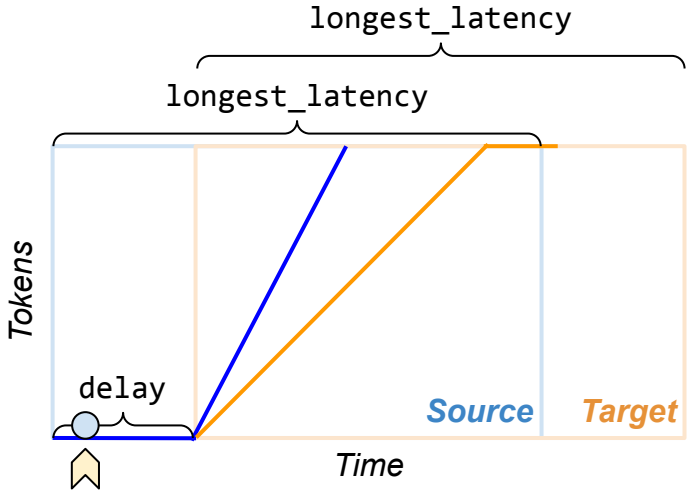
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

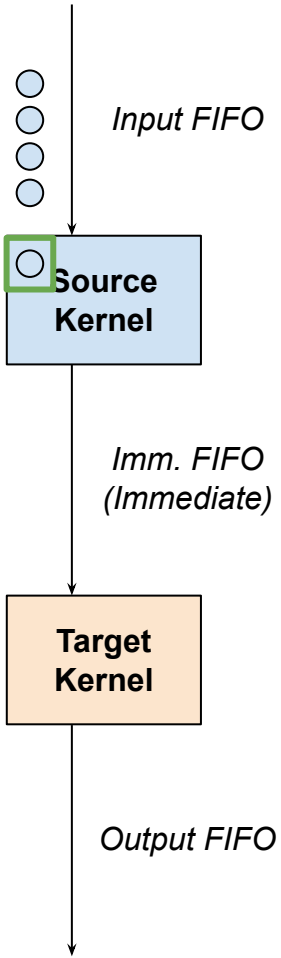




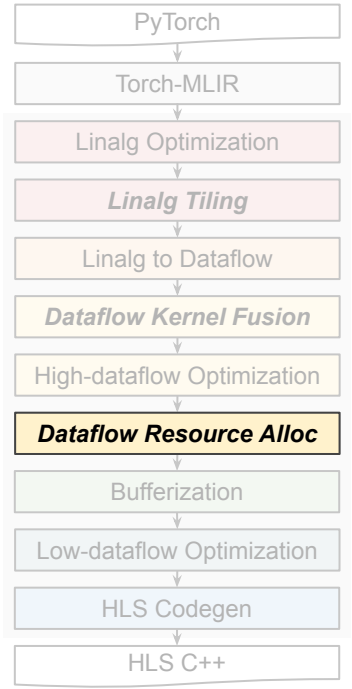
Overall Token Status Curve (Imm. FIFO)

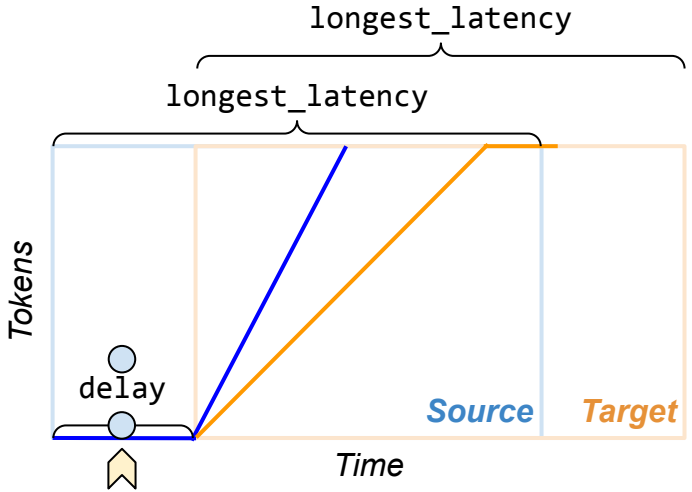
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

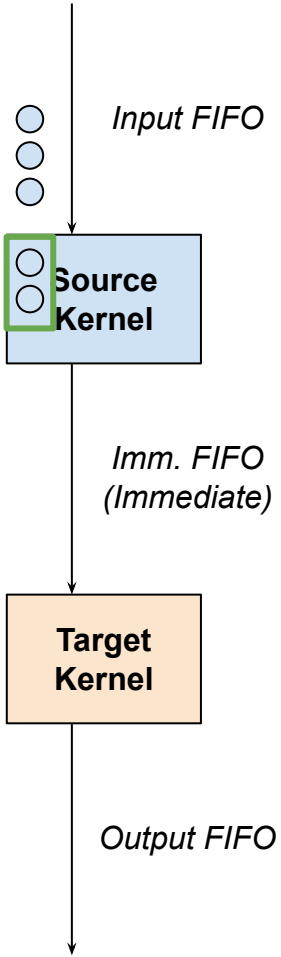




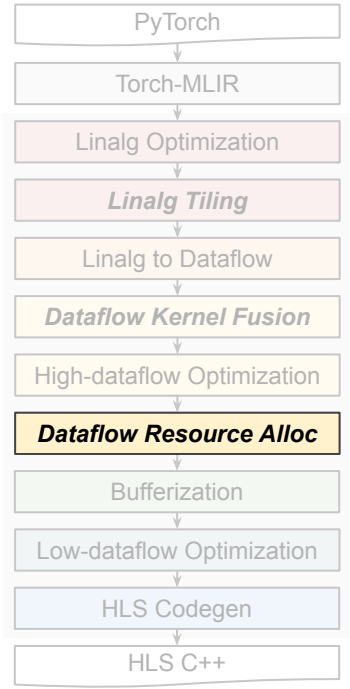
Overall Token Status Curve (Imm. FIFO)

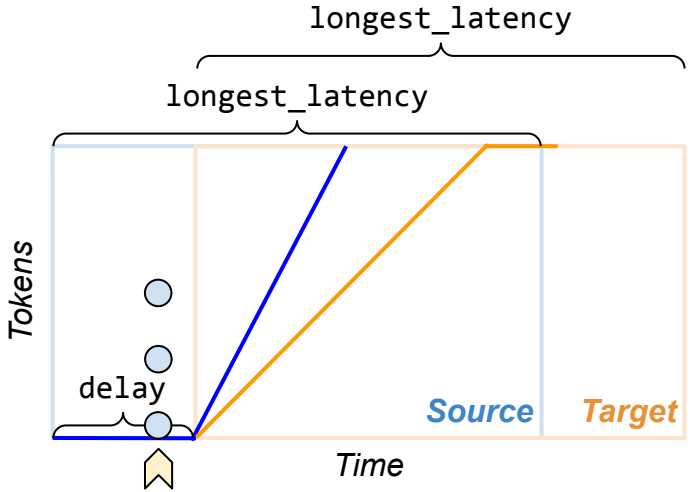
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

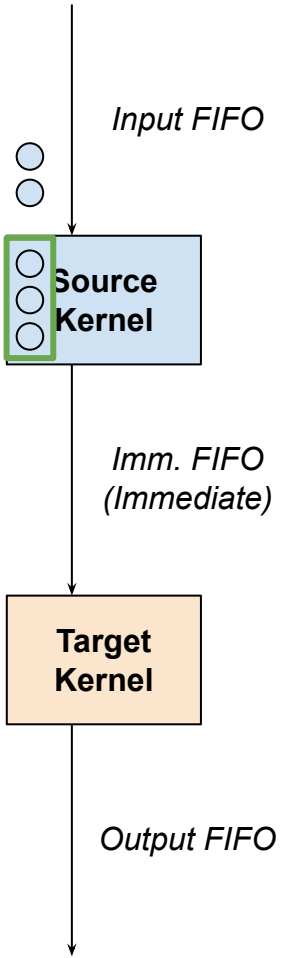




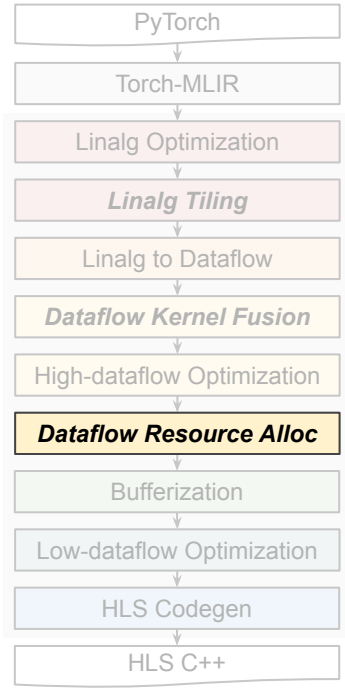
Overall Token Status Curve (Imm. FIFO)

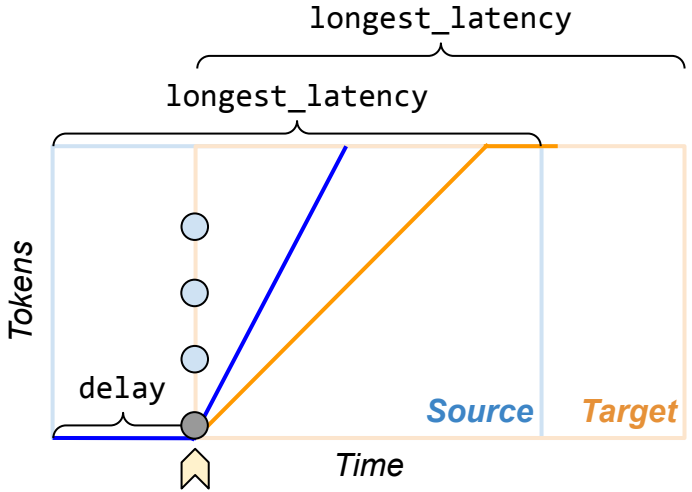
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

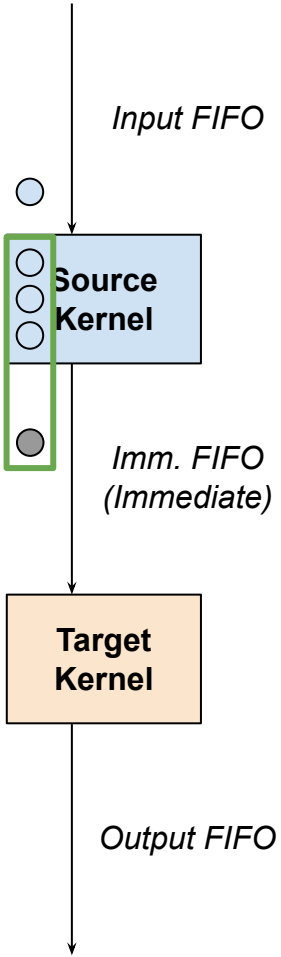




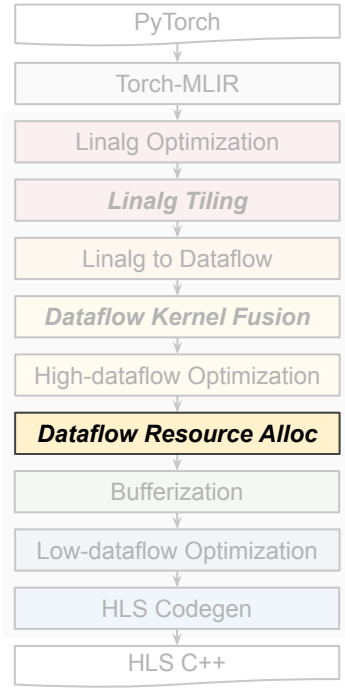
Overall Token Status Curve (Imm. FIFO)

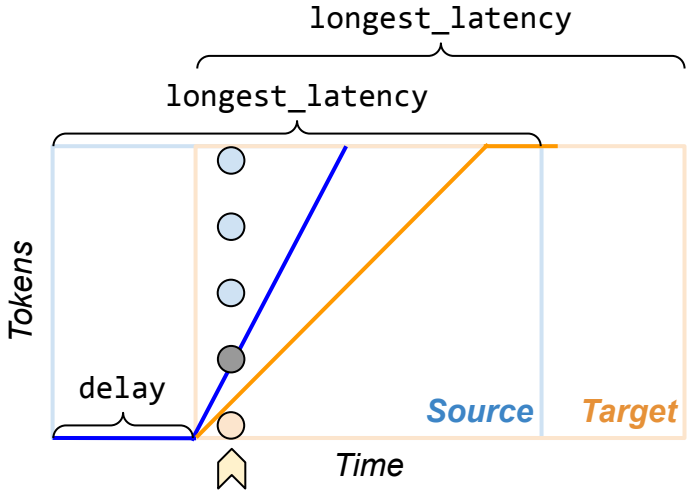
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

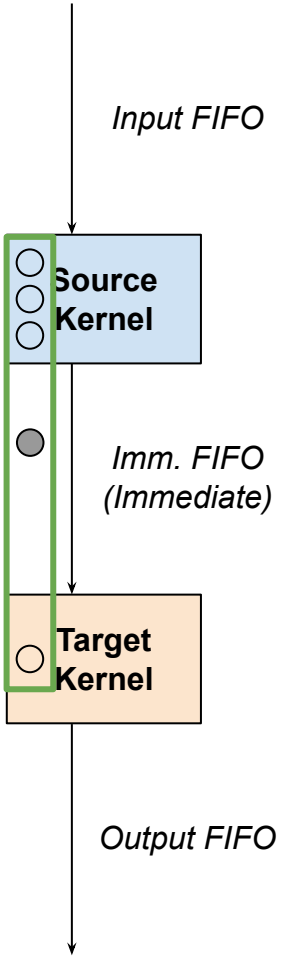




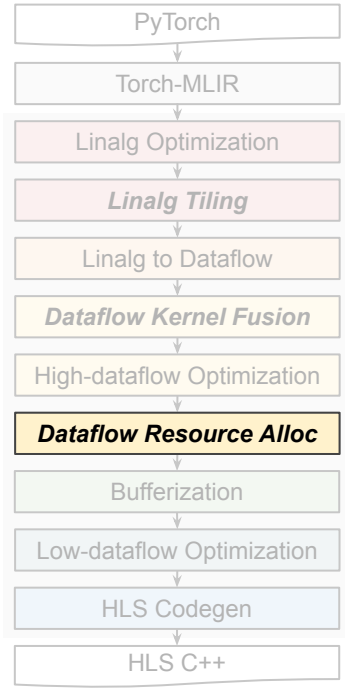
Overall Token Status Curve (Imm. FIFO)

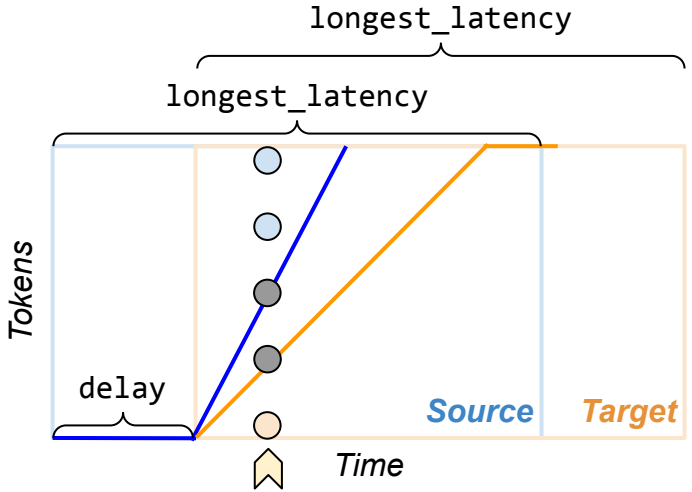
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

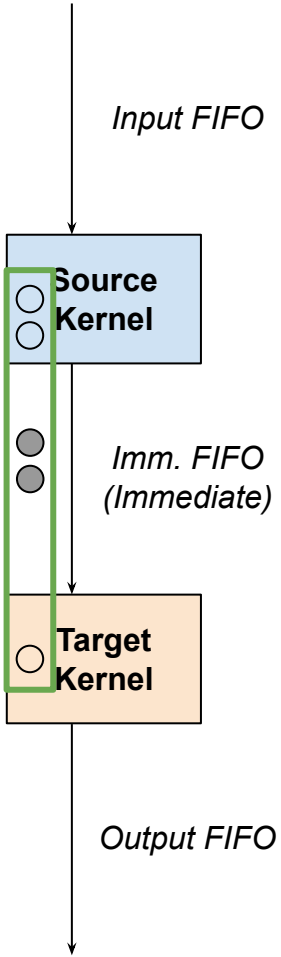




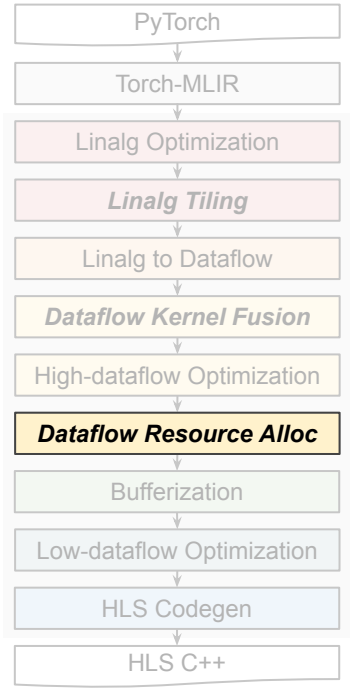
Overall Token Status Curve (Imm. FIFO)

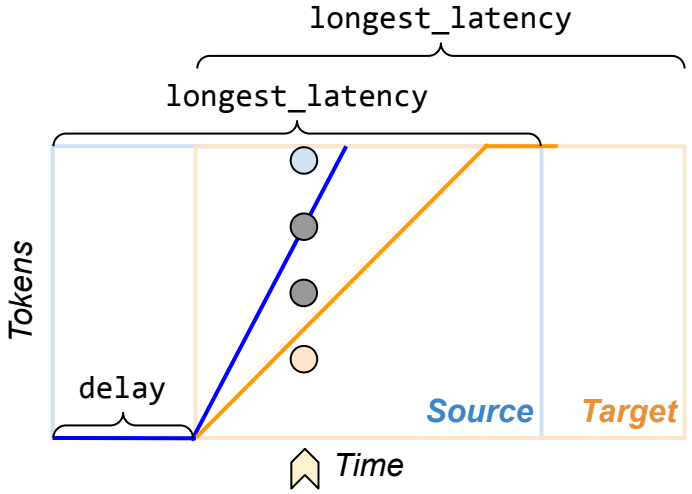
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

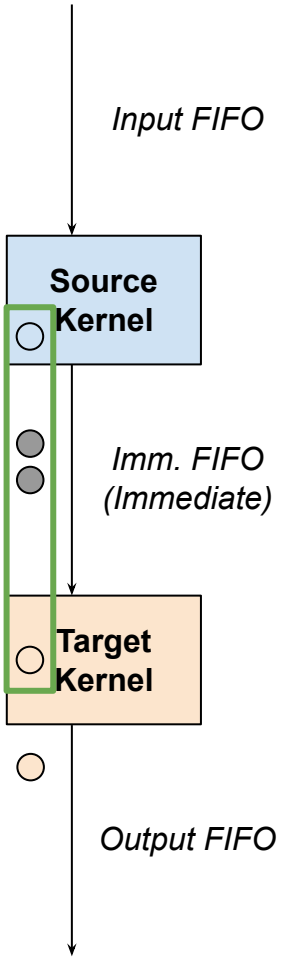




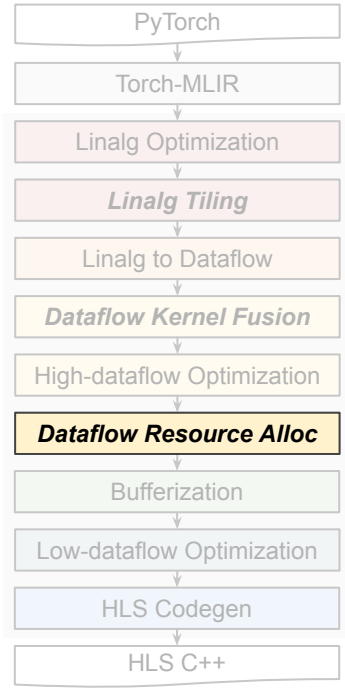
Overall Token Status Curve (Imm. FIFO)

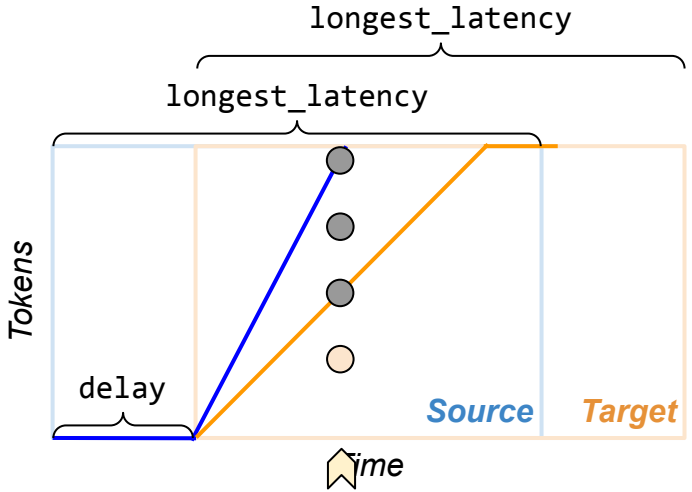
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

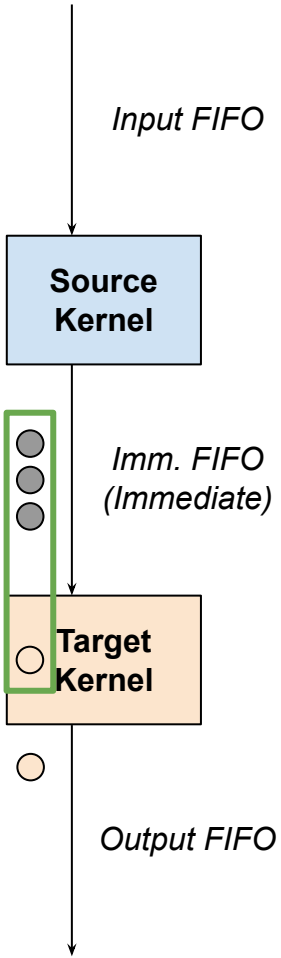




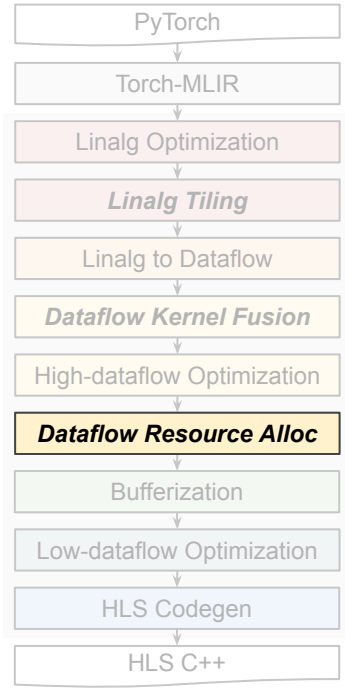
Overall Token Status Curve (Imm. FIFO)

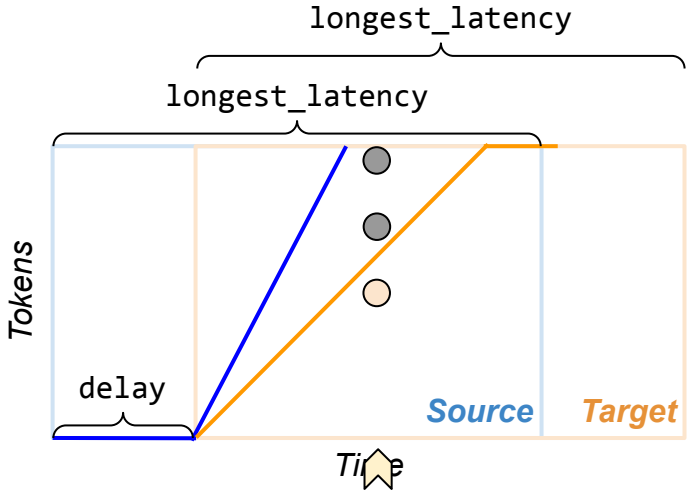
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

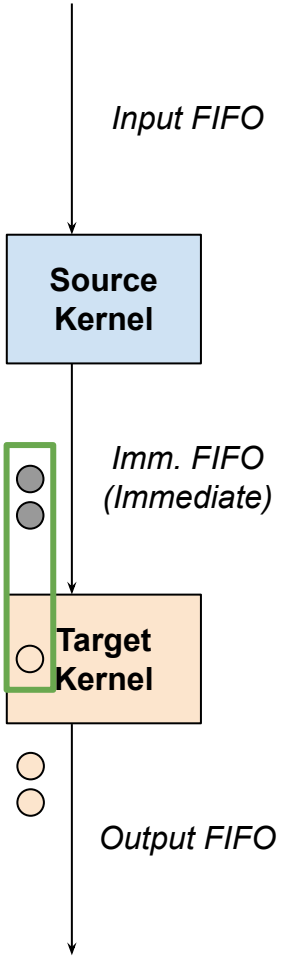




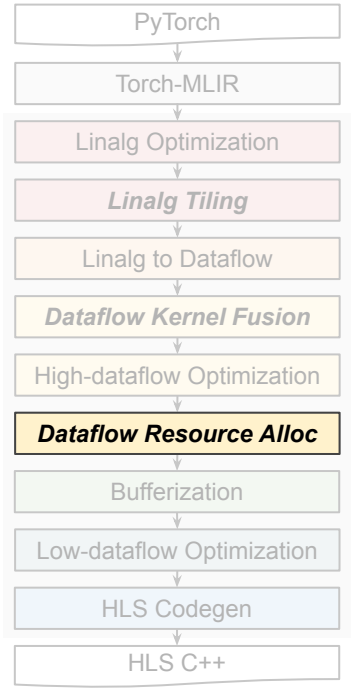
Overall Token Status Curve (Imm. FIFO)

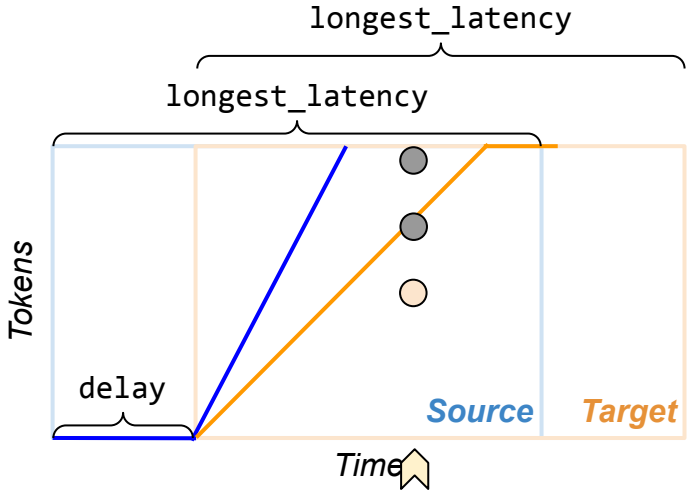
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

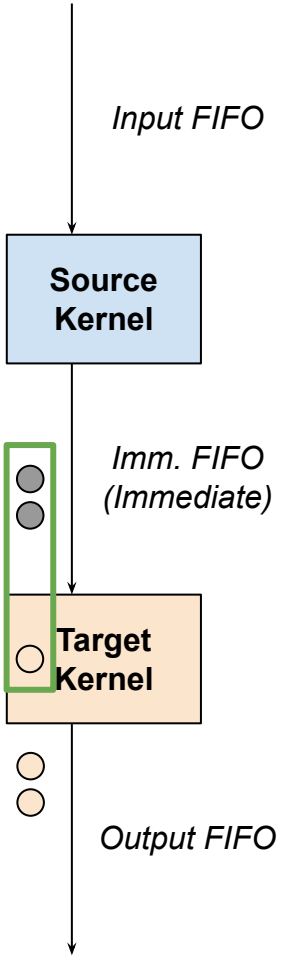




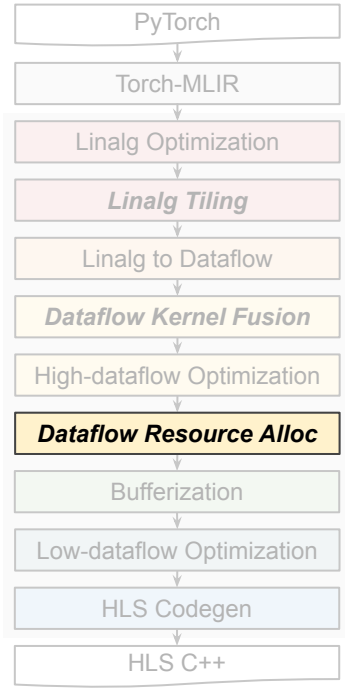
Overall Token Status Curve (Imm. FIFO)

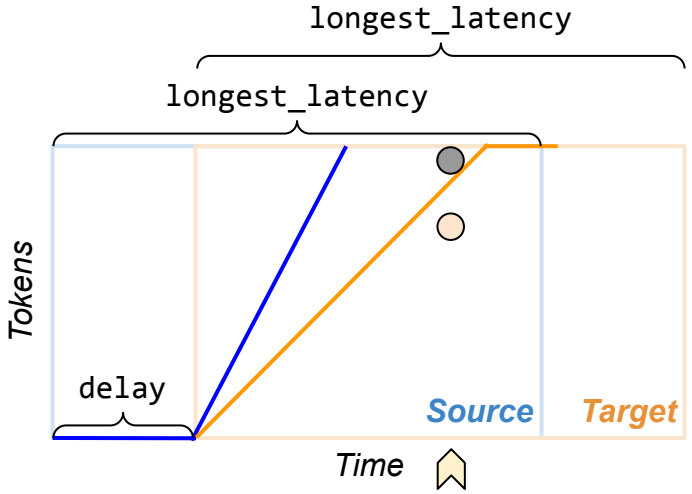
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

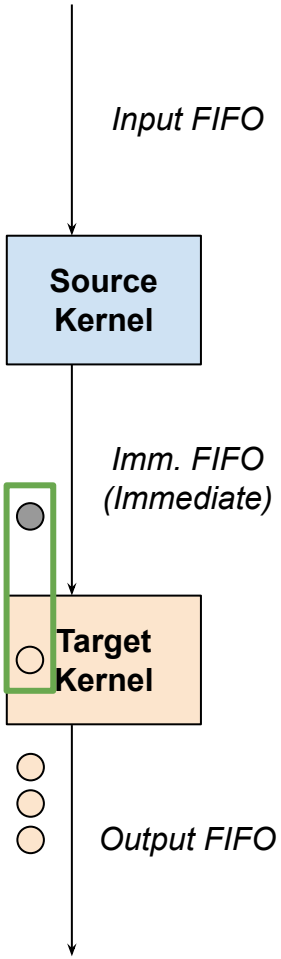




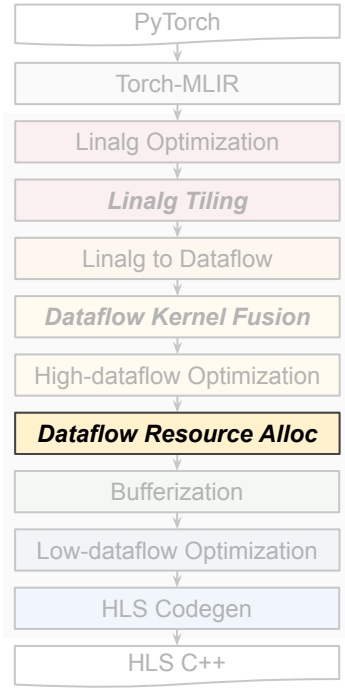
Overall Token Status Curve (Imm. FIFO)

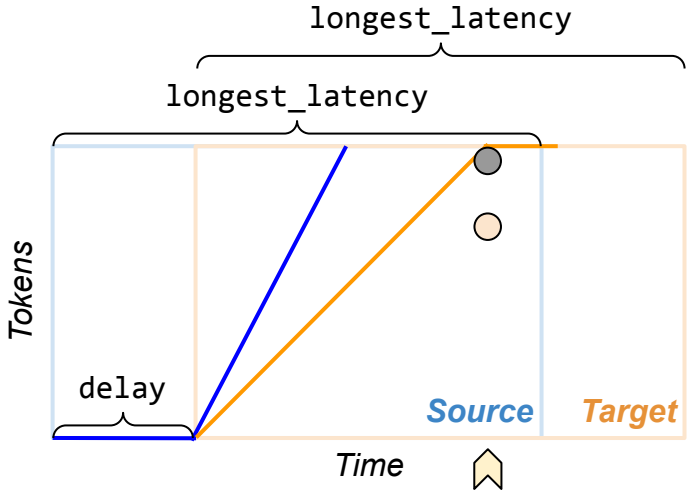
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

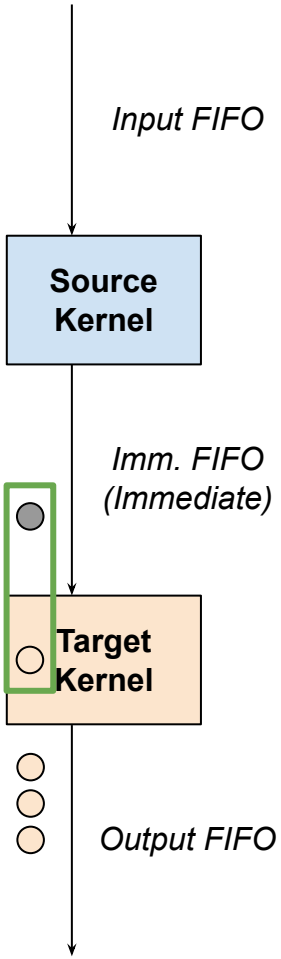




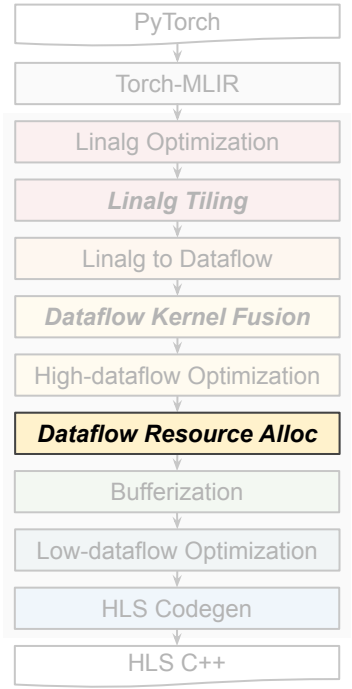
Overall Token Status Curve (Imm. FIFO)

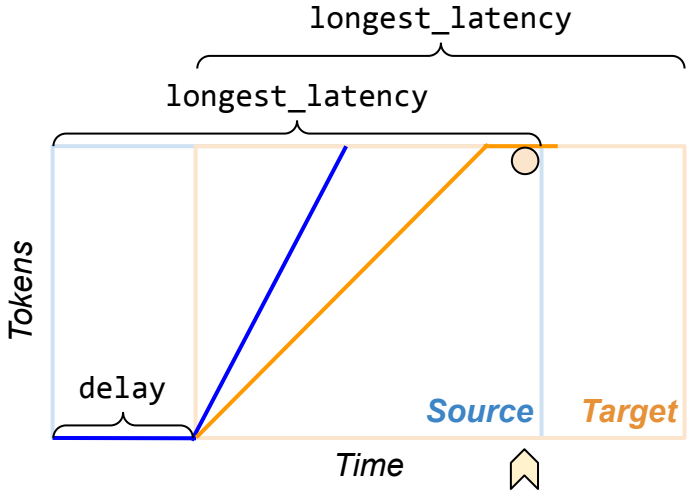
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

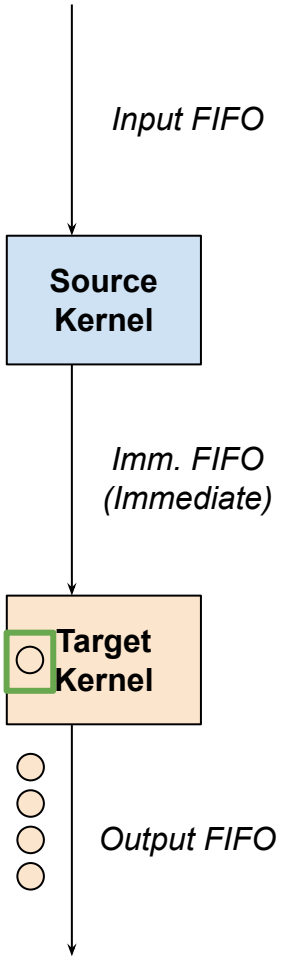




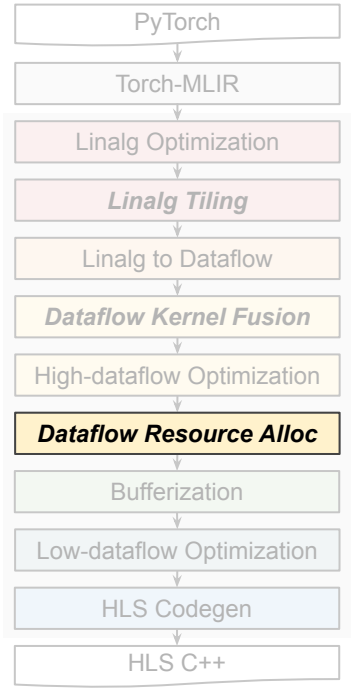
Overall Token Status Curve (Imm. FIFO)

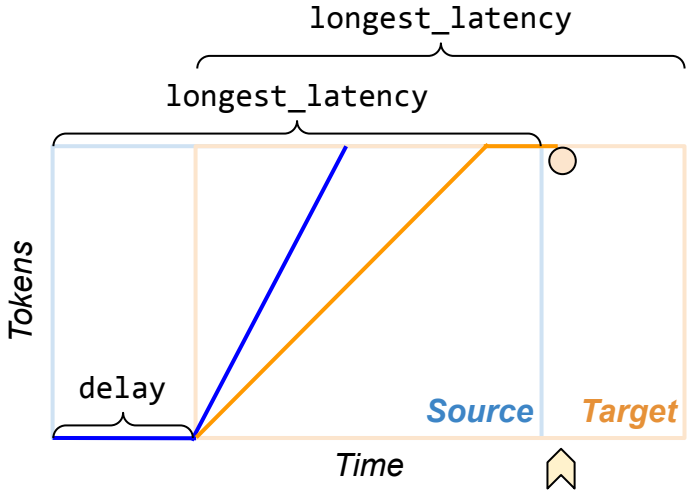
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

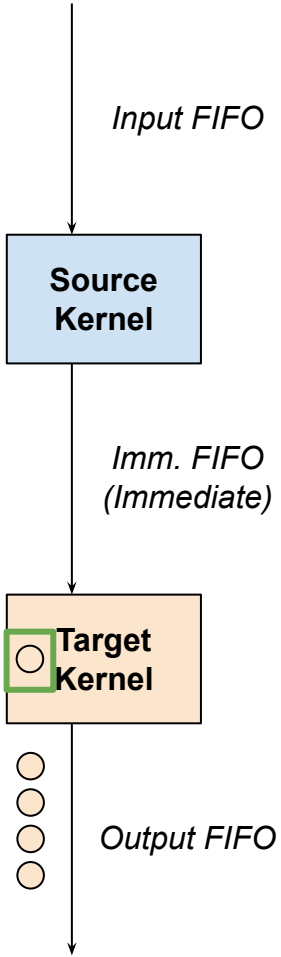




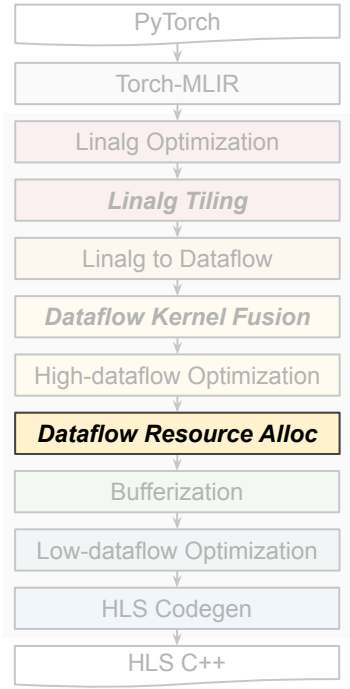
Overall Token Status Curve (Imm. FIFO)

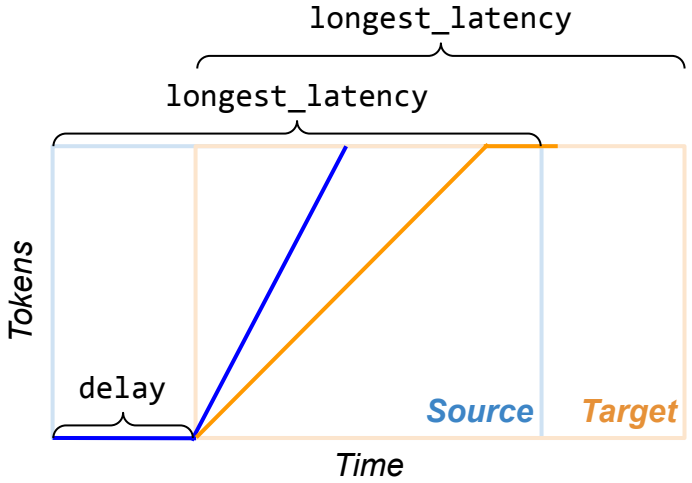
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

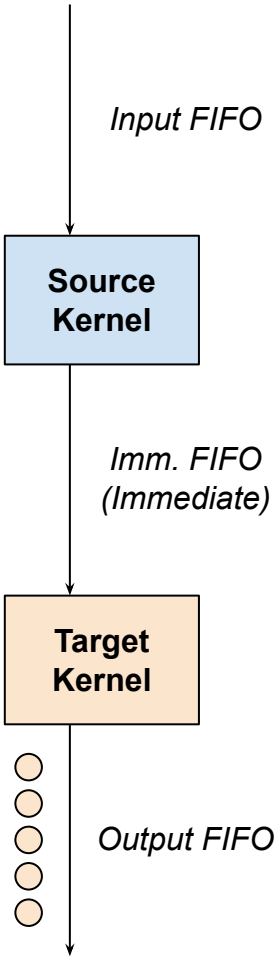




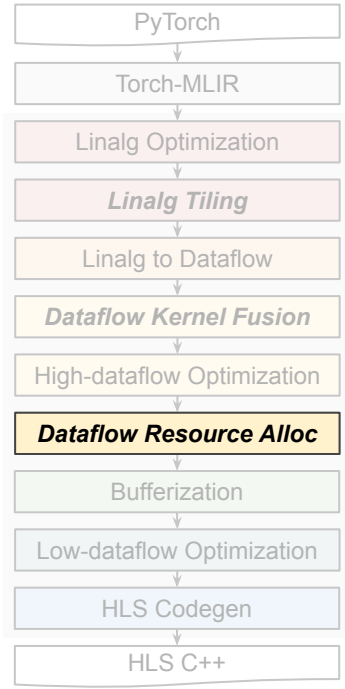
Overall Token Status Curve (Imm. FIFO)

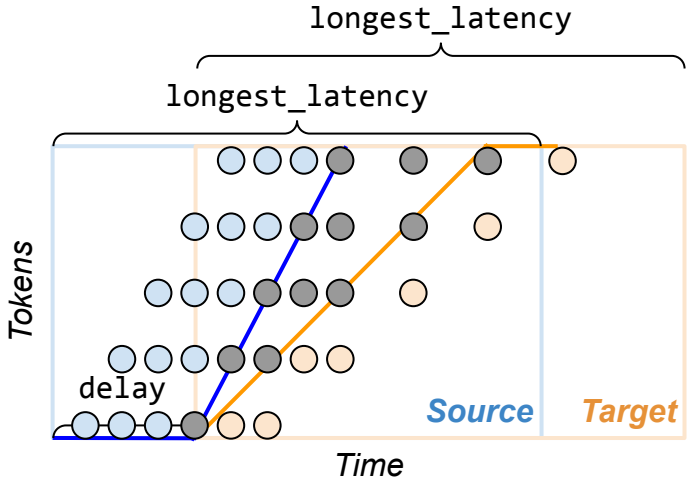
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Number Estimation Heuristic (Cont.)

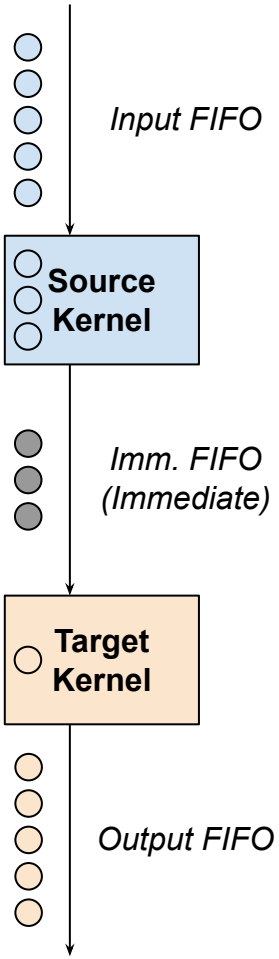




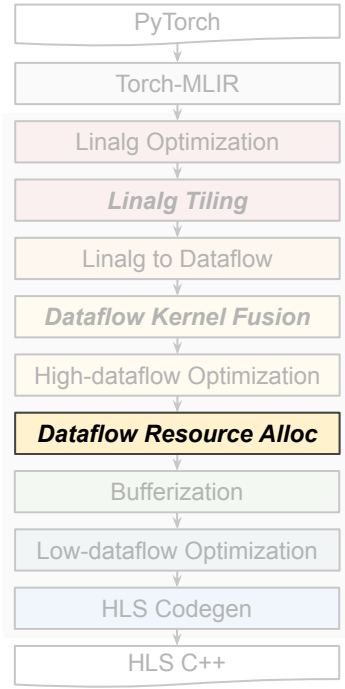
Overall Token Status Curve (Imm. FIFO)

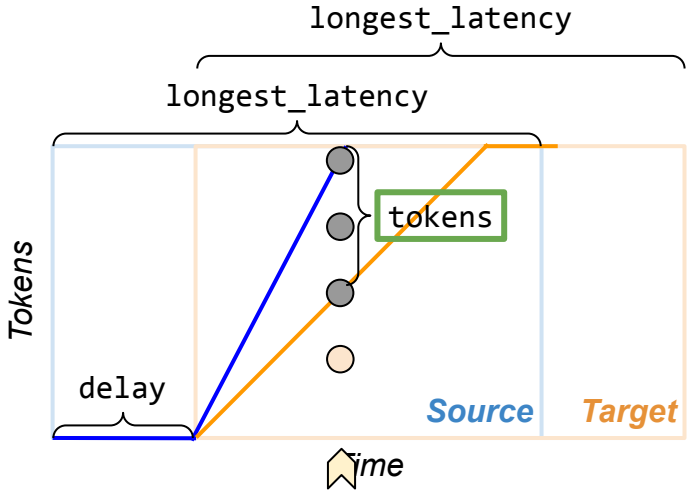
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



Token Behavior Modeling (Cont.)

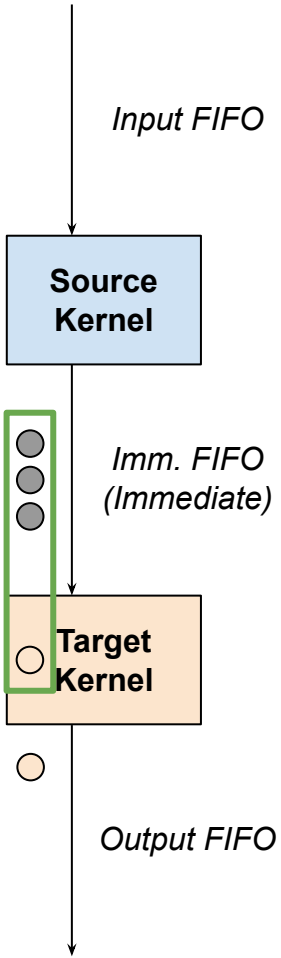




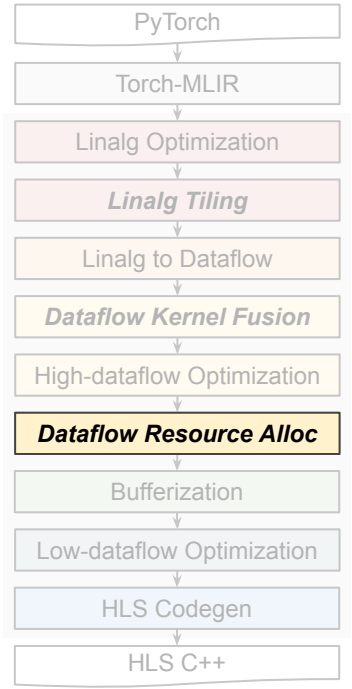
Overall Token Status Curve (Imm. FIFO)

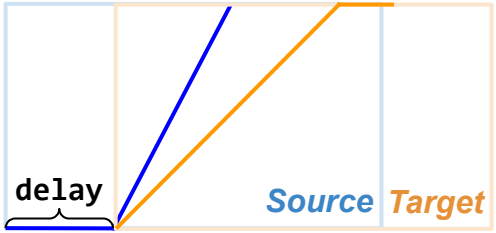
`token` = Atomic element passed between dataflow kernels

`longest_latency` = The longest latency among all the dataflow kernels in the accelerator, determining the maximum throughput



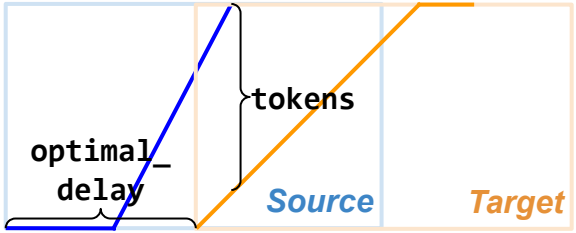
Linear Programming (LP) Formulation





(1) Calculate token production/consumption curves based on kernel profiling results

(2) Calculate minimum `delay` given a strategy

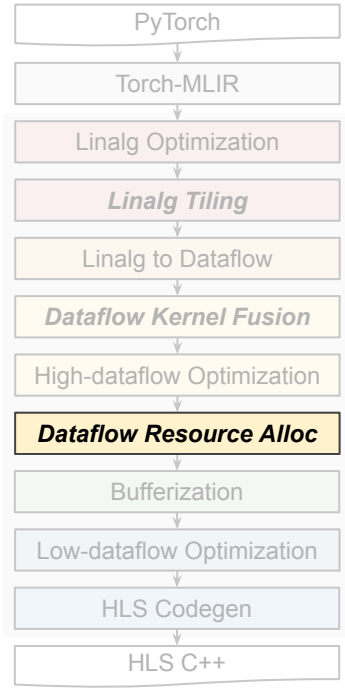


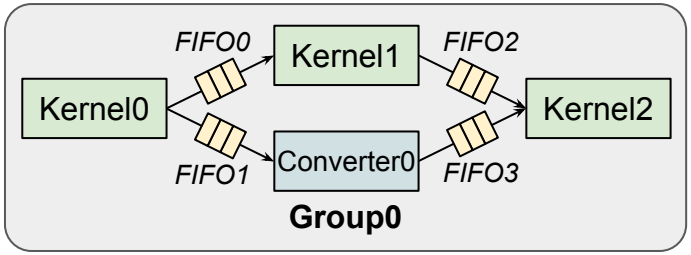
(3) Solve `optimal_delay` with LP

(4) Calculate `tokens` based on `optimal_delay` given a strategy

Transform FIFO sizing into a scheduling problem

Linear Programming (LP) Formulation (Cont.)





Delay Variables

- `var["fifo0"]`
- `var[“fifo1”]`
- `var[“fifo2”]`
- `var[“fifo3”]`

Minimum `delay` Constraints

- `var["fifo0"] >= delay["kernel0"]`
- `var[“fifo1”] >= delay[“kernel0”]`
- `var[“fifo2”] >= delay[“kernel1”]`
- `var[“fifo3”] >= delay[“Converter0”]`

Objective

- `obj = var["fifo0"] + var[“fifo1”] + var[“fifo2”] + var[“fifo3”]`
- `minimize obj`

Path Balance Constraints

- `var["fifo0"] + var[“fifo2”] >= critical_delay[“kernel0”][“kernel2”]`
- `var["fifo1"] + var[“fifo3”] >= critical_delay[“kernel0”][“kernel2”]`

StreamTensor Outline

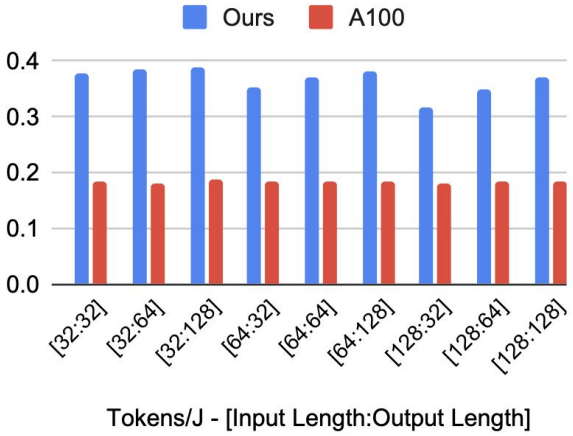
- Motivation
- StreamTensor Typing System
- StreamTensor Compilation Pipeline
- StreamTensor Design Spaces
- StreamTensor Results

FPGA On-board Results on GPT-2

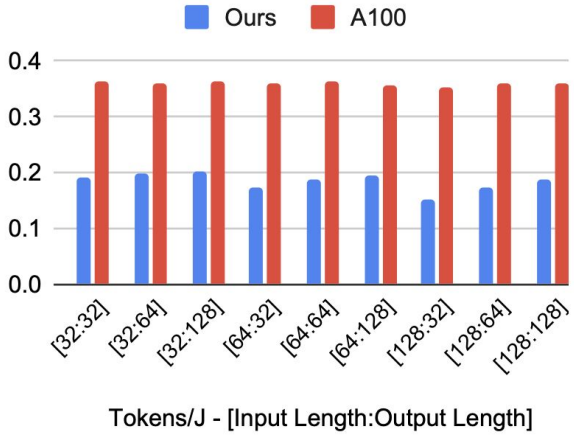
[Input Len: Output Len]	Ours			Allo [15] (Ratio of $\frac{Ours}{Allo}$)			DFX [29] (Ratio of $\frac{Ours}{DFX}$)		
	Latency (ms)	TTFT (ms)	Speed (token/s)	Latency (ms)	TTFT (ms)	Speed (token/s)	Latency (ms)	TTFT (ms)	Speed (token/s)
[32:32]	194.99	34.59	199.51	238.32 (0.82x)	81.50 (0.42x)	204.05 (0.98x)	350.00 (0.56x)	177.20 (0.20x)	185.19 (1.08x)
[64:64]	358.24	61.27	215.51	476.64 (0.75x)	162.99 (0.38x)	204.05 (1.06x)	694.70 (0.52x)	349.10 (0.18x)	185.19 (1.16x)
[128:128]	696.65	125.35	224.05	953.28 (0.73x)	325.98 (0.38x)	204.05 (1.10x)	1384.00 (0.50x)	692.80 (0.18x)	185.19 (1.21x)
[256:256]	1387.76	272.85	229.61	1906.56 (0.73x)	651.96 (0.42x)	204.05 (1.13x)	2800.00 (0.50x)	1417.60 (0.19x)	185.19 (1.24x)
Geo. Mean	-	-	-	**0.76x**	**0.40x**	**1.06x**	**0.52x**	**0.19x**	**1.17x**

[Input Len: Output Len]	Ours			A100 (Ratio of $\frac{Ours}{A100}$)			2080Ti (Ratio of $\frac{Ours}{2080Ti}$)		
	Latency (ms)	TTFT (ms)	Speed (token/s)	Latency (ms)	TTFT (ms)	Speed (token/s)	Latency (ms)	TTFT (ms)	Speed (token/s)
[32:32]	194.99	34.59	199.51	291.16 (0.67x)	8.72 (3.97x)	113.30 (1.76x)	518.46 (0.38x)	24.98 (1.38x)	64.85 (3.08x)
[64:64]	358.24	61.27	215.51	567.41 (0.63x)	8.76 (6.99x)	114.56 (1.88x)	1010.81 (0.35x)	25.23 (2.43x)	64.94 (3.32x)
[128:128]	696.65	125.35	224.05	1118.28 (0.62x)	8.65 (14.49x)	115.35 (1.94x)	3969.76 (0.18x)	25.26 (4.96x)	32.45 (6.90x)
[256:256]	1387.76	272.85	229.61	2227.79 (0.62x)	8.53 (31.99x)	115.35 (1.99x)	7914.23 (0.18x)	25.23 (10.81x)	32.45 (7.08x)
Geo. Mean	-	-	-	**0.64x**	10.65x	**1.89x**	**0.25x**	3.67x	**4.73x**

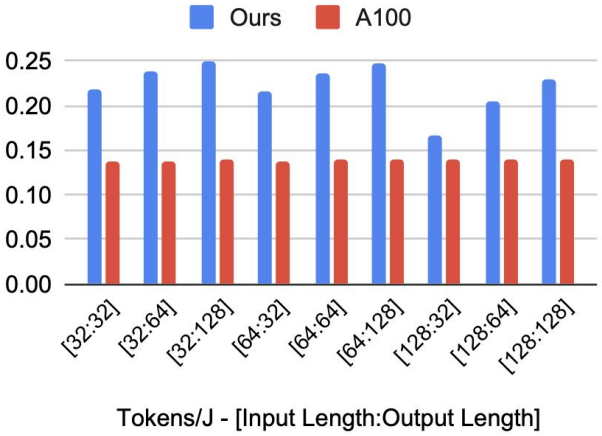
FPGA On-board Results on Emerging LLMs



(a) Qwen.



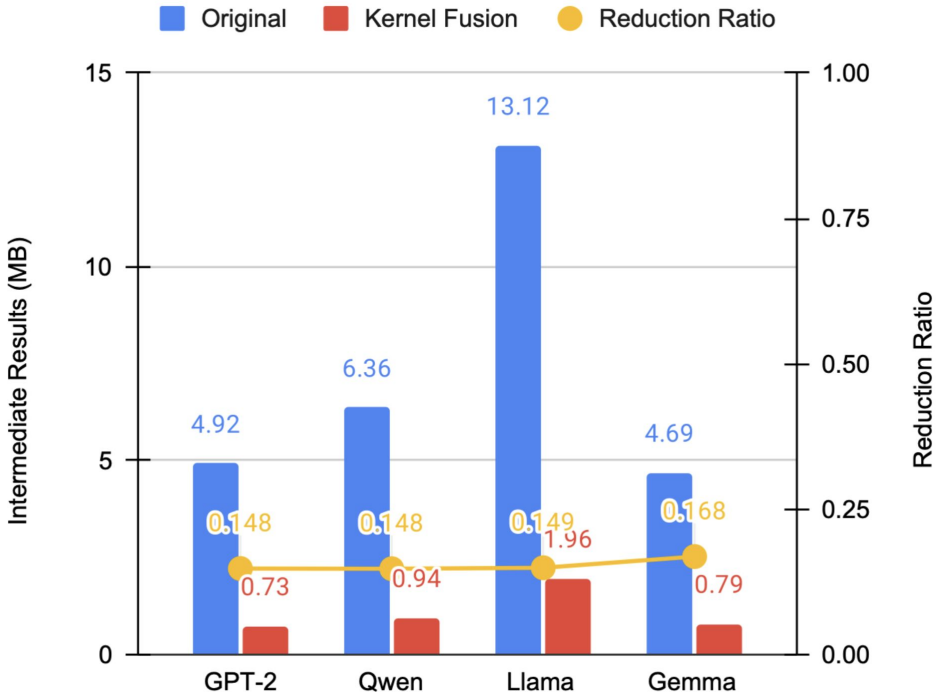
(b) Llama.



(c) Gemma.

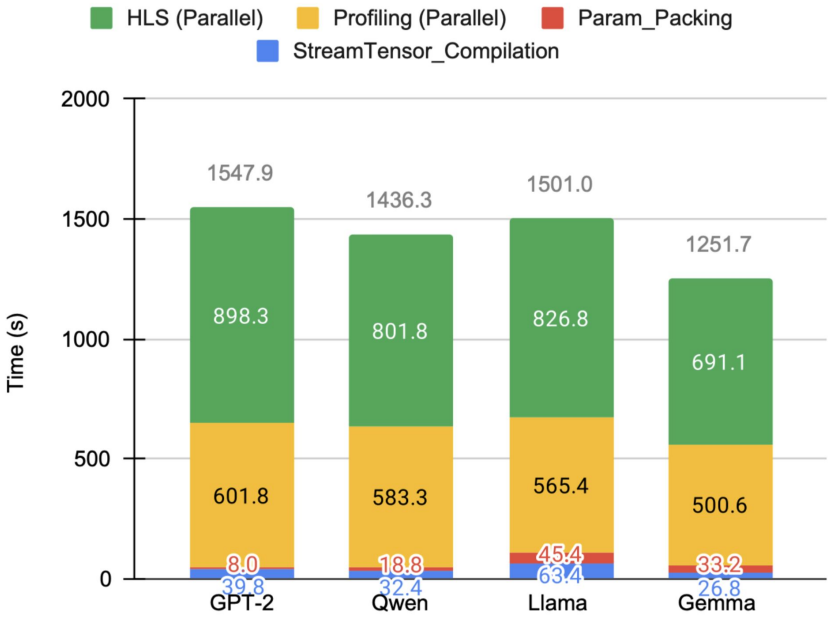
- Achieved higher energy efficiency on Qwen and Gemma.
- Energy efficiency of Llama is lower than GPU because the intermediate results of Llama is larger than other models, limiting the design space explorations.

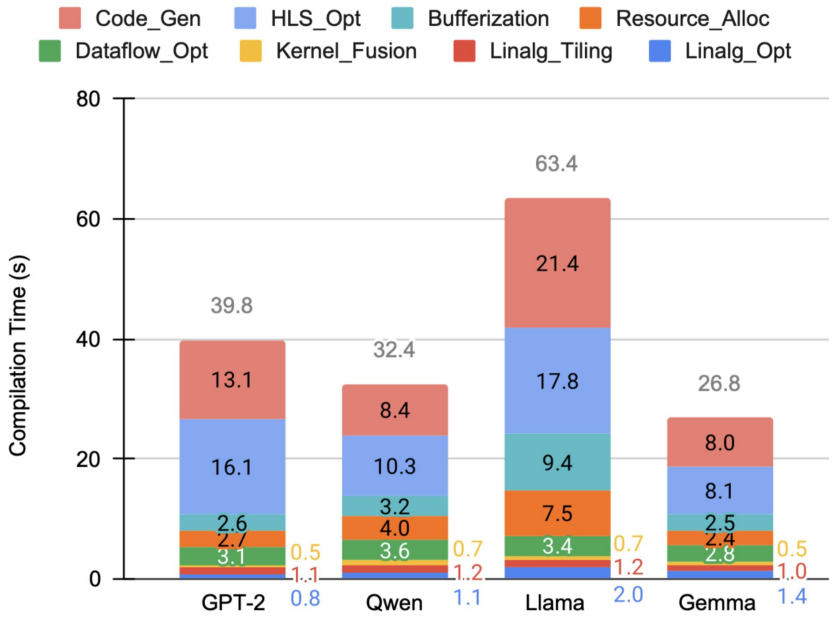
On-chip Memory Reduction through Kernel Fusion



- Only consider intermediate results, i.e., activations, in this study.
- Parameters are always stored in external memory.
- On-chip memory are reduced to 0.15x - 0.17x through stream-based kernel fusion.
- Without kernel fusion, the on-chip memory resources are not enough to store all intermediate results

RTL Generation Time





- For RTL generation, the downstream HLS synthesis and profiling consume most execution time
- StreamTensor compilation and parameter packing only consume a tiny portion of execution time
- For StreamTensor compilation, the low-level transforms (bufferization, HLS opt., and codegen) consume much more execution than high-level transforms (Linalg tiling, kernel fusion, dataflow opt.), showing the efficiency of the proposed itensor-based dataflow optimizations.

Thanks!

Hanchen Ye, ElastixAI
hanchenye@gmail.com
March 10, 2025