



# Tensor Evolution

Javed Absar, Principal Engineer  
Muthu M. Basakaran, Principal Engineer

UK Reservoir Labs R&D  
Qualcomm Technologies International, Ltd.



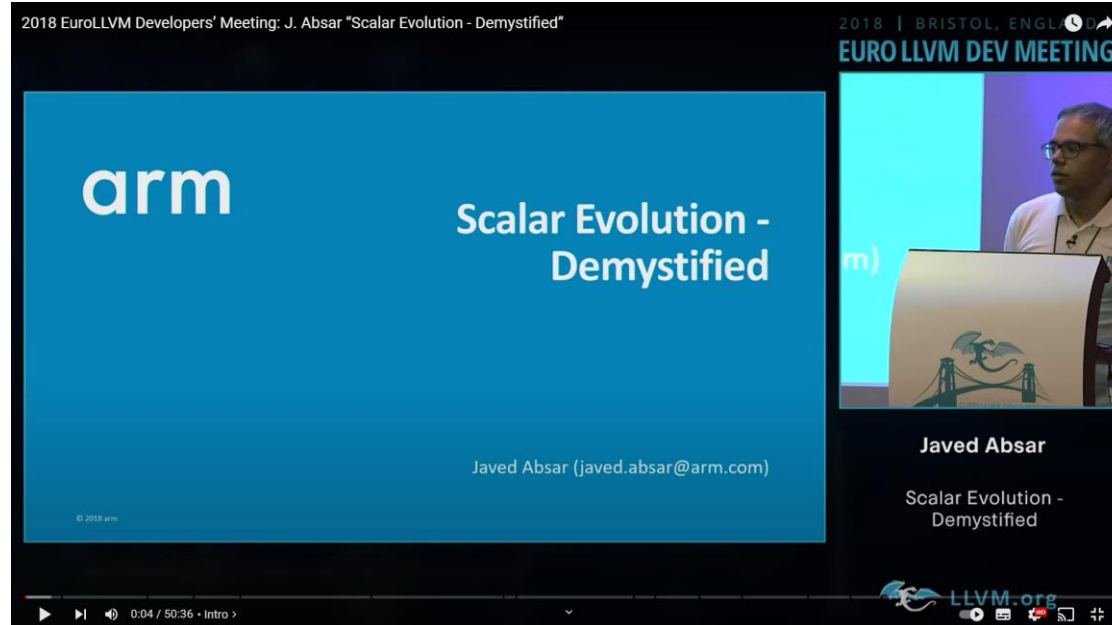
# Tensor Evolution

- Extension of LLVM Scalar Evolution (SCEV) for Tensors
  - Analysis and Optimization Technique
- Tensors are
  - multi-dimensional arrays
  - fundamental to Machine Learning models

# Scalar Evolution (SCEV)

*“Scalar Evolution is an LLVM analysis that is used to analyze, categorize and simplify expressions in loops. Many optimizations such as - generalized loop-strength-reduction, parallelization by induction variable (vectorization), and loop-invariant expression elimination - rely on **SCEV** analysis. However, SCEV is also a complex topic.”*

-- some Large Language Model



2018 EuroLLVM Developers' Meeting: J. Absar "Scalar Evolution - Demystified"

2018 | BRISTOL, ENGLAND  
EURO LLVM DEV MEETING

arm

Scalar Evolution - Demystified

Javed Absar (javed.absar@arm.com)

© 2018 arm

Javed Absar

Scalar Evolution - Demystified

LLVM.org

## The LLVM Compiler Infrastructure

### 2018 European LLVM Developers Meeting

#### Scalar Evolution - Demystified

J. Absar

This is a tutorial/technical-talk proposal for an illustrative and in-depth exposition of Scalar Evolution in LLVM. Scalar Evolution is an L analyse, categorize and simplify expressions in loops. Many optimizations such as - generalized loop-strength-reduction, parallelisation b (vectorization), and loop-invariant expression elimination - rely on SCEV analysis.

However, **SCEV is also a complex topic**. This tutorial delves into how exactly LLVM performs the SCEV magic and how it can be used to analyse different optimisations.

This tutorial will cover the following topics:

1. What is SCEV? How does it help improve performance? SCEV in action (using simple clear examples).
2. Chain of Recurrences - which forms the mathematical basis of SCEV.
3. Simplifying/rewriting rules in CR that SCEV uses to simplify expressions evolving out of induction variables. Terminology and SC AddRec) that is common currency that one should get familiar with when trying to understand and use SCEV in any context.
4. LLVM SCEV implementation of CR - what's present and what's missing?
5. How to use SCEV analysis to write your own optimisation pass? Usage of SCEV by LSR (Loop Strength Reduce) and others.
6. How to generate analysis info out of SCEV and how to interpret them.

The last talk on SCEV was in LLVM-Dev 2009. This tutorial will be complementary to that and go further with examples, discussions and llvm since then. The author has previously given a talk on machine scheduler in llvm - <https://www.youtube.com/watch?v=brpomKUynE>

**BoFs (Birds of a Feather)**

# Scalar Evolution

- SCEV analysis and opt

```
int foo(int *a, int n, int k){  
  for (int i = 0; i < n; i++)  
    a[i] = i*k;  
}
```

```
$ opt -analyze -scalar-evolution foo.ll
```

1. Printing analysis 'Scalar Evolution Analysis' for function 'foo':
2. Classifying expressions for: @foo
3. ...
4. %mul = mul nsw i32 %i, %k
5. --> {0,+,%k}<%for.body> Exits: ((-1 + %n) \* %k)
6. ...

# Tensor Evolution – Motivating Example 1

```
# PyTorch code.  
# a and x are tensors  
def forward(self, a, x):  
    for _ in range(15):  
        x = a + x  
    return x
```

- Tensor Evolution Optimization

```
# PyTorch code.  
# a and x are tensors  
def forward(self, a, x):  
    return 15*a+x
```

# Mathematical Formulation

- Basic Recurrence (Tensor Evolution)
  - a constant or loop-invariant tensor  $T_c$
  - a function  $\tau_1$  over natural number  $N$  that produces tensor of same shape as  $T_c$
  - an element-wise operator  $+$  associative and commutative
  - $\tau$  defined as function  $\tau(i)$  over  $N$

$$\tau = \{ T_c, +, \tau_1 \} \quad \text{eq. 1}$$

$$\{ T_c, +, \tau_1 \}(i) = T_c + \tau_1(0) + \tau_1(1) \dots + \tau_1(i - 1) \quad \text{eq. 2}$$

# Mathematical Formulation

- Chain of Recurrences (Tensor Evolution)
  - loop-invariant tensors  $Tc_0, Tc_1, Tc_2, \dots, Tc_{i-1}$  ;
  - function  $\tau_k$  defined over  $N$ ,
  - operators  $\odot_1, \odot_2, \dots, \odot_k$ ,
  - chain of evolution of tensor value represented by tuple

$$\tau = \{Tc_0, \odot_1, Tc_1, \odot_2, \dots, \odot_k, \tau_k\} \quad \text{eq. 1}$$

$$\tau(i) = \{Tc_0, \odot_1, \{Tc_1, \odot_2, \dots, \odot_k, \tau_k\}\}(i) \quad \text{eq. 2}$$

- Note: Operators could be same or different (+, -, \*, tanh).
- Recurrences
  - Algebraic properties
  - Computationally reducible at any iteration point

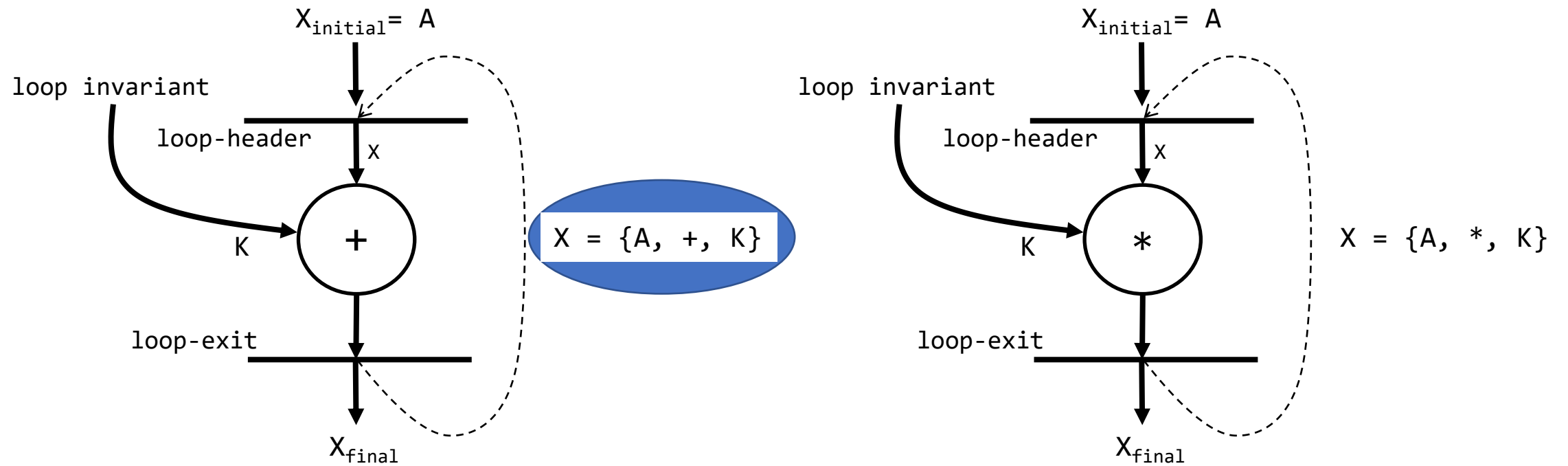
# Tensor Evolution

- Lemmas – Rewrite Rules
- Used for building TEV ‘available’ expressions and simplifications

operator	TEV expression	rewrite rule
slice	$\text{slice}(\{A, +, \tau\})$	$\{\text{slice}(A), +, \text{slice}(\tau)\}$
	$\text{slice}(\{A, *, \tau\})$	$\{\text{slice}(A), *, \text{slice}(\tau)\}$
reshape	$\text{reshape}(\{A, \odot, \tau\})$	$\{\text{reshape}(A), \odot, \text{reshape}(\tau)\}$
concat	$\text{concat}(\{A, \odot, \tau_1\}, \{B, \odot, \tau_2\})$	$\{\text{concat}(A, B), \odot, \text{concat}(\tau_1, \tau_2)\}$
add K	$K + \{A, +, \tau\}$	$\{K+A, +, \tau\}$
add TEVs	$\{A, +, \tau_1\} + \{B, +, \tau_2\}$	$\{A+B, +, \tau_1+\tau_2\}$
mul	$K * \{A, +, \tau\}$	$\{K*A, +, K*\tau\}$
inject TEV	$\{A, +, \{B, +, \tau\}\}$	$\{A, +, B, +, \tau\}$

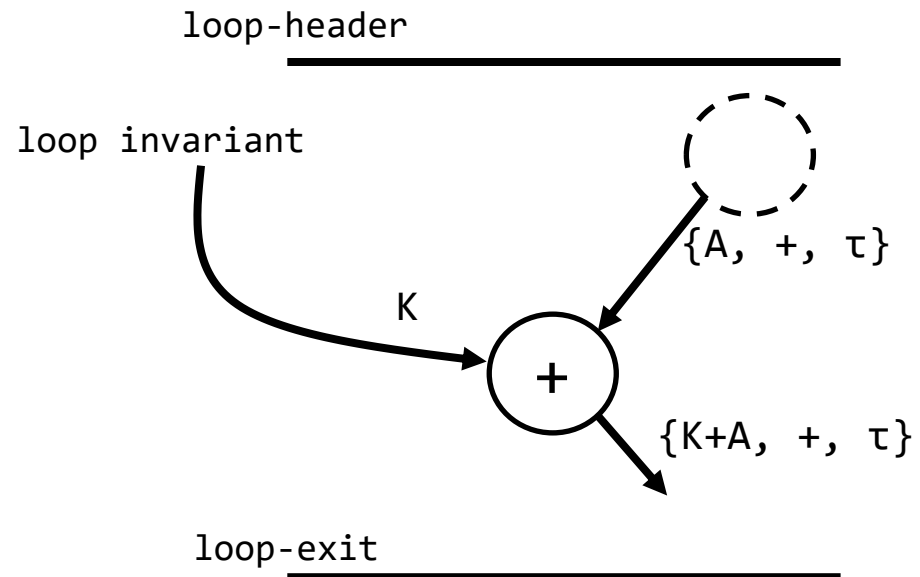


# Tensor Evolution – Basic Recurrence



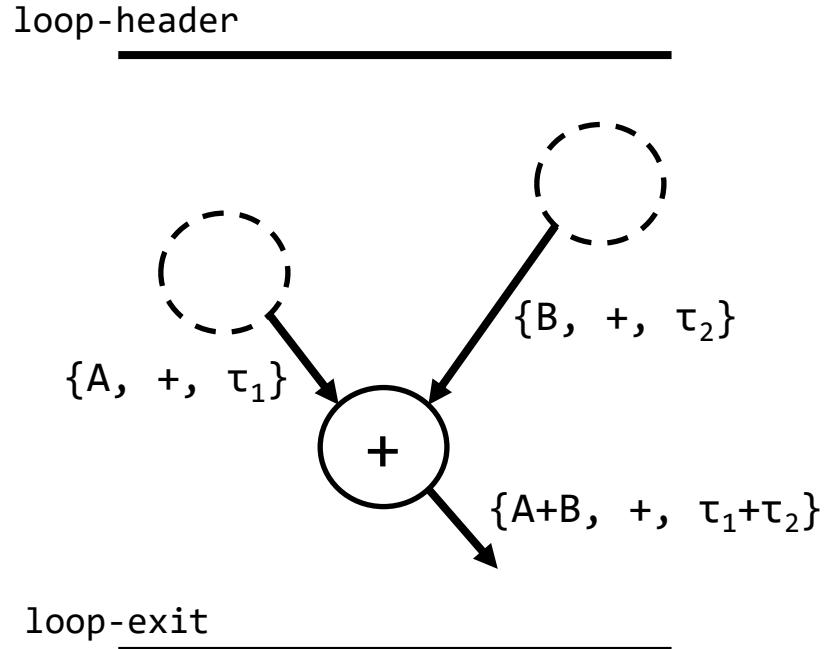
# Tensor Evolution

- Lemma: Add a constant (LIV) tensor



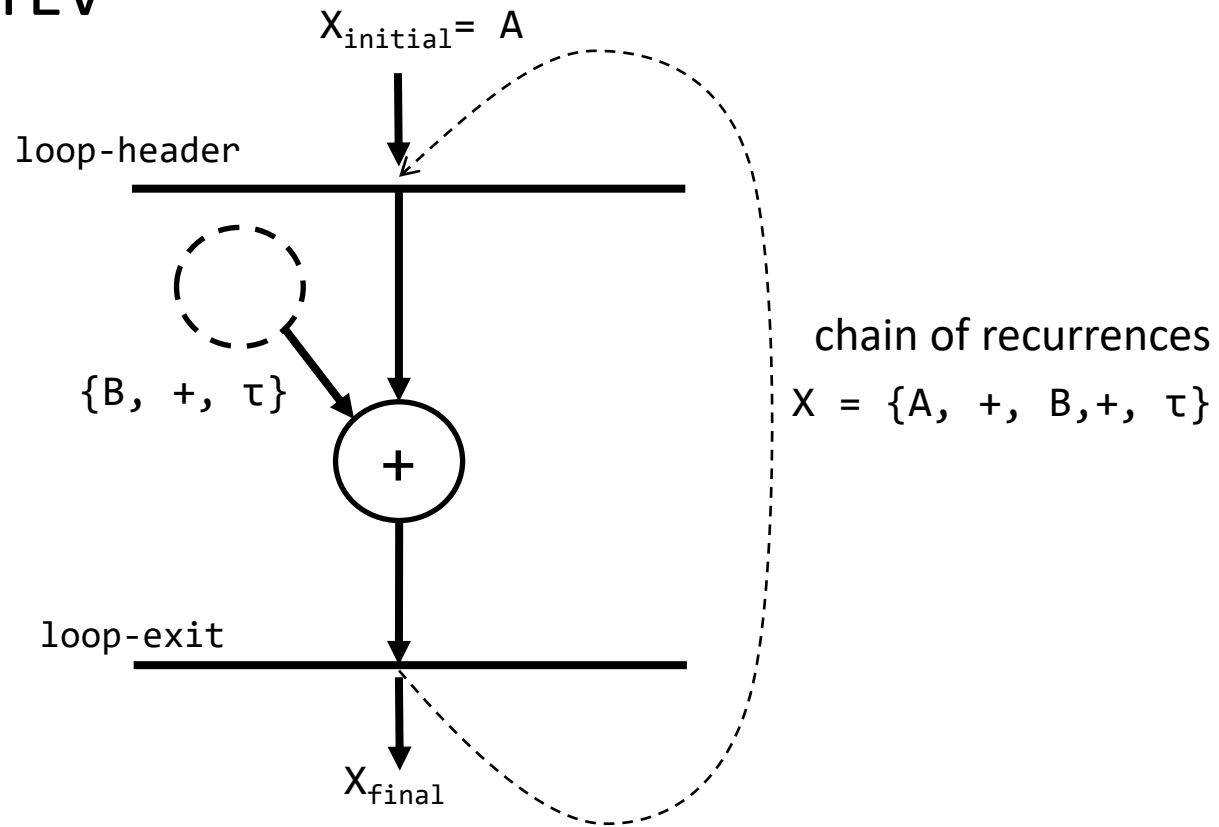
# Tensor Evolution

- Lemma: Add two TEVs



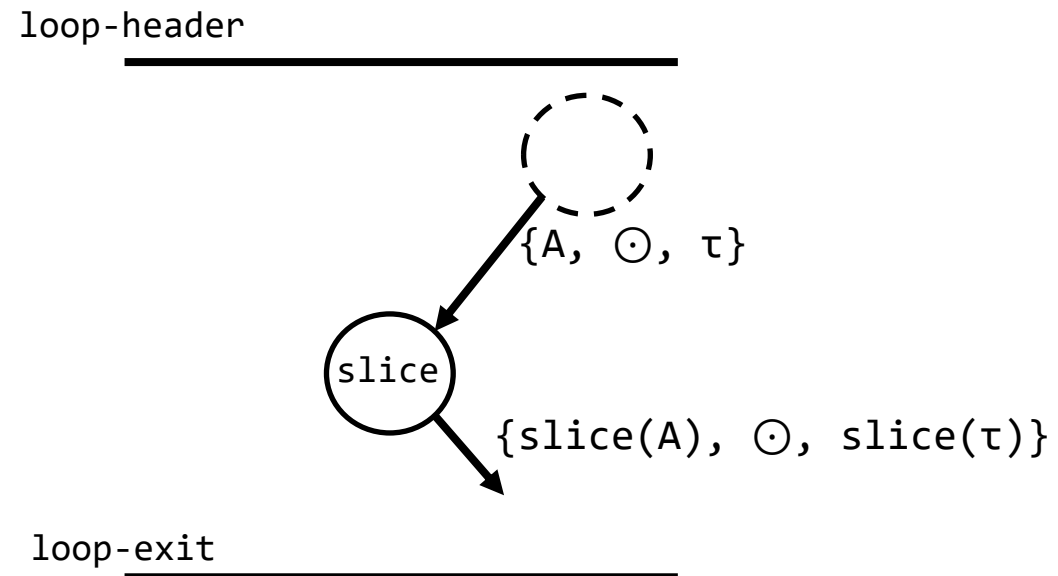
# Tensor Evolution

- Lemma: TEV inject into TEV



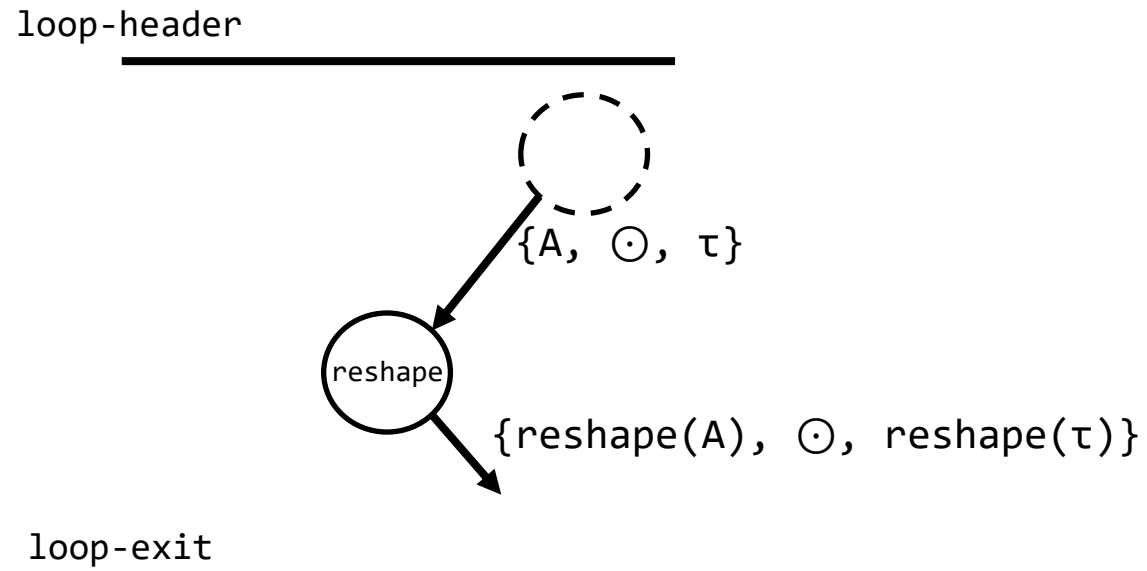
# Tensor Evolution

- Lemma: Slice



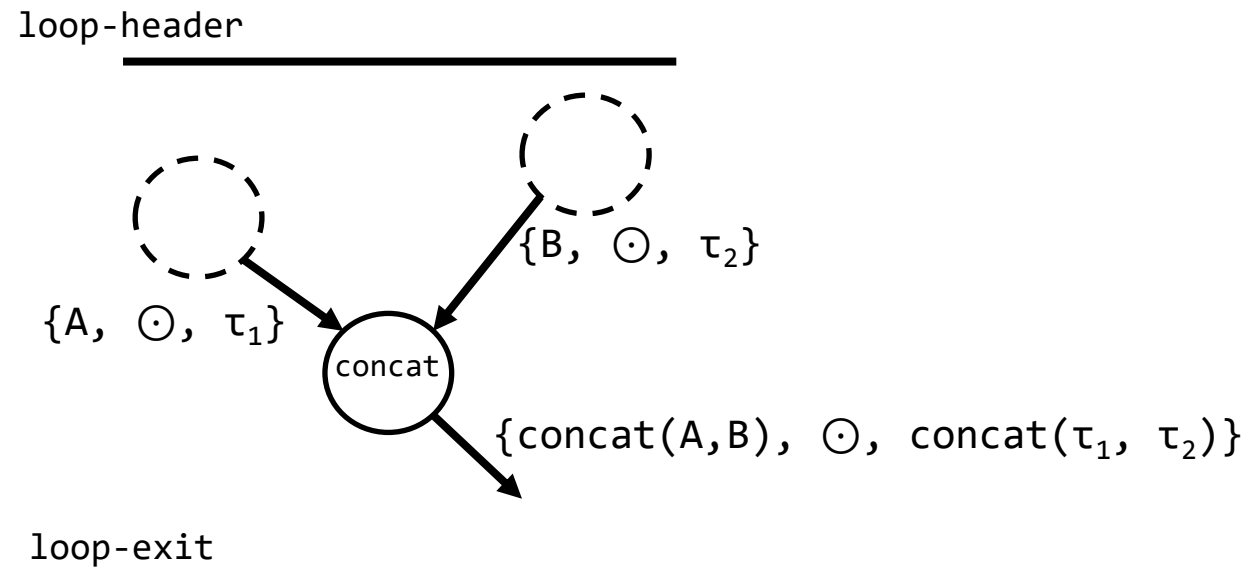
# Tensor Evolution

- Lemma: Reshape



# Tensor Evolution

- Lemma: Concat



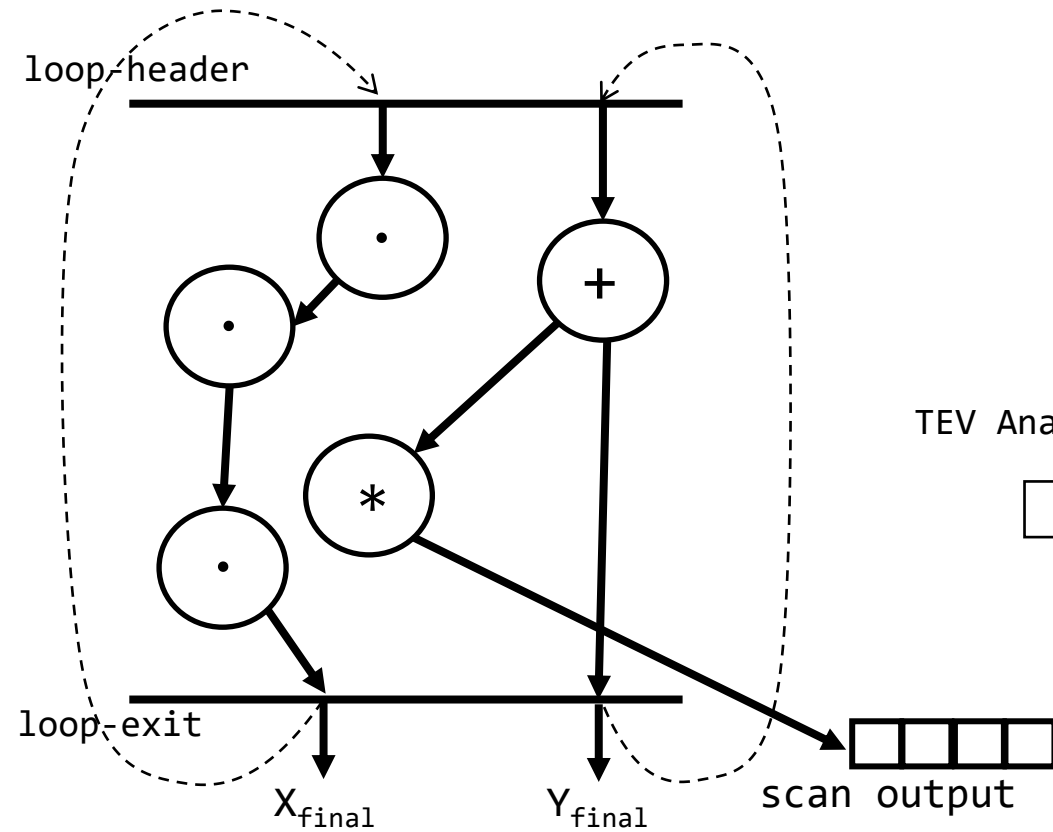
# Tensor Evolution

- Lemmas – Rewrite Rules
- Used for building TEV expressions and simplifications

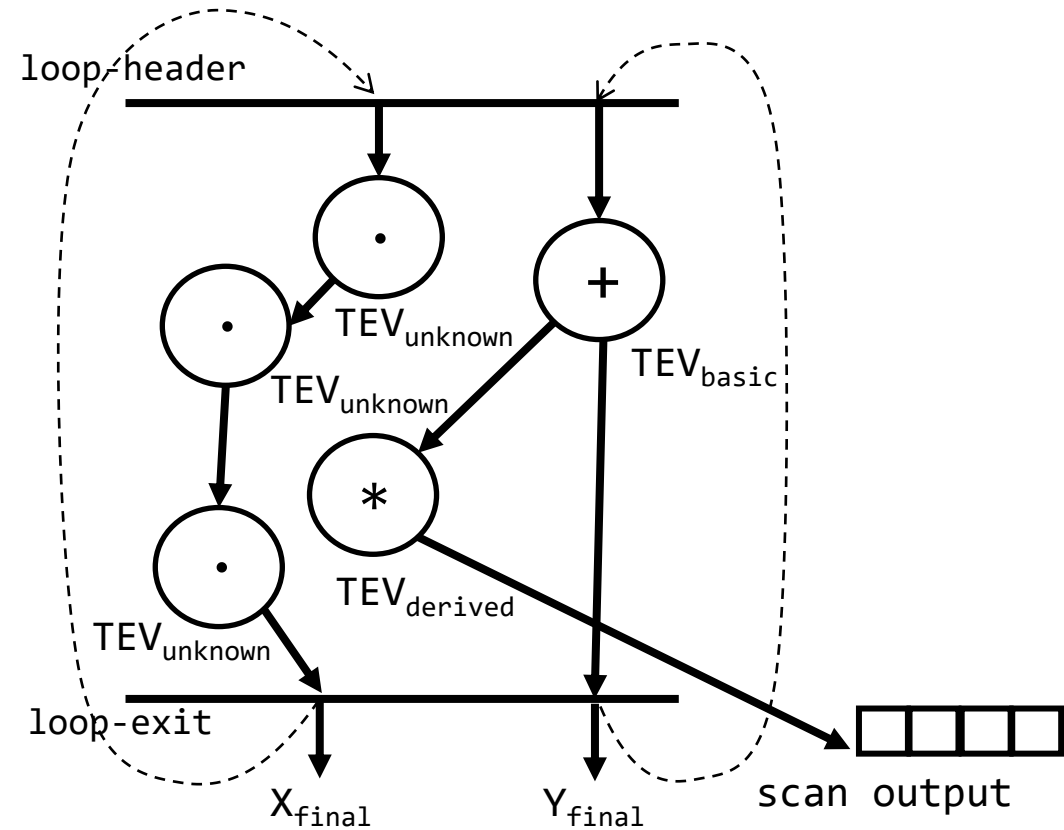
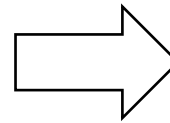
operator	TEV expression	rewrite rule
slice	$\text{slice}(\{A, +, \tau\})$	$\{\text{slice}(A), +, \text{slice}(\tau)\}$
	$\text{slice}(\{A, *, \tau\})$	$\{\text{slice}(A), *, \text{slice}(\tau)\}$
reshape	$\text{reshape}(\{A, \odot, \tau\})$	$\{\text{reshape}(A), \odot, \text{reshape}(\tau)\}$
concat	$\text{concat}(\{A, \odot, \tau_1\}, \{B, \odot, \tau_2\})$	$\{\text{concat}(A, B), \odot, \text{concat}(\tau_1, \tau_2)\}$
add K	$K + \{A, +, \tau\}$	$\{K+A, +, \tau\}$
add TEVs	$\{A, +, \tau_1\} + \{B, +, \tau_2\}$	$\{A+B, +, \tau_1+\tau_2\}$
mul	$K * \{A, +, \tau\}$	$\{K*A, +, K*\tau\}$
inject TEV	$\{A, +, \{B, +, \tau\}\}$	$\{A, +, B, +, \tau\}$



# TEV Pass - Analysis

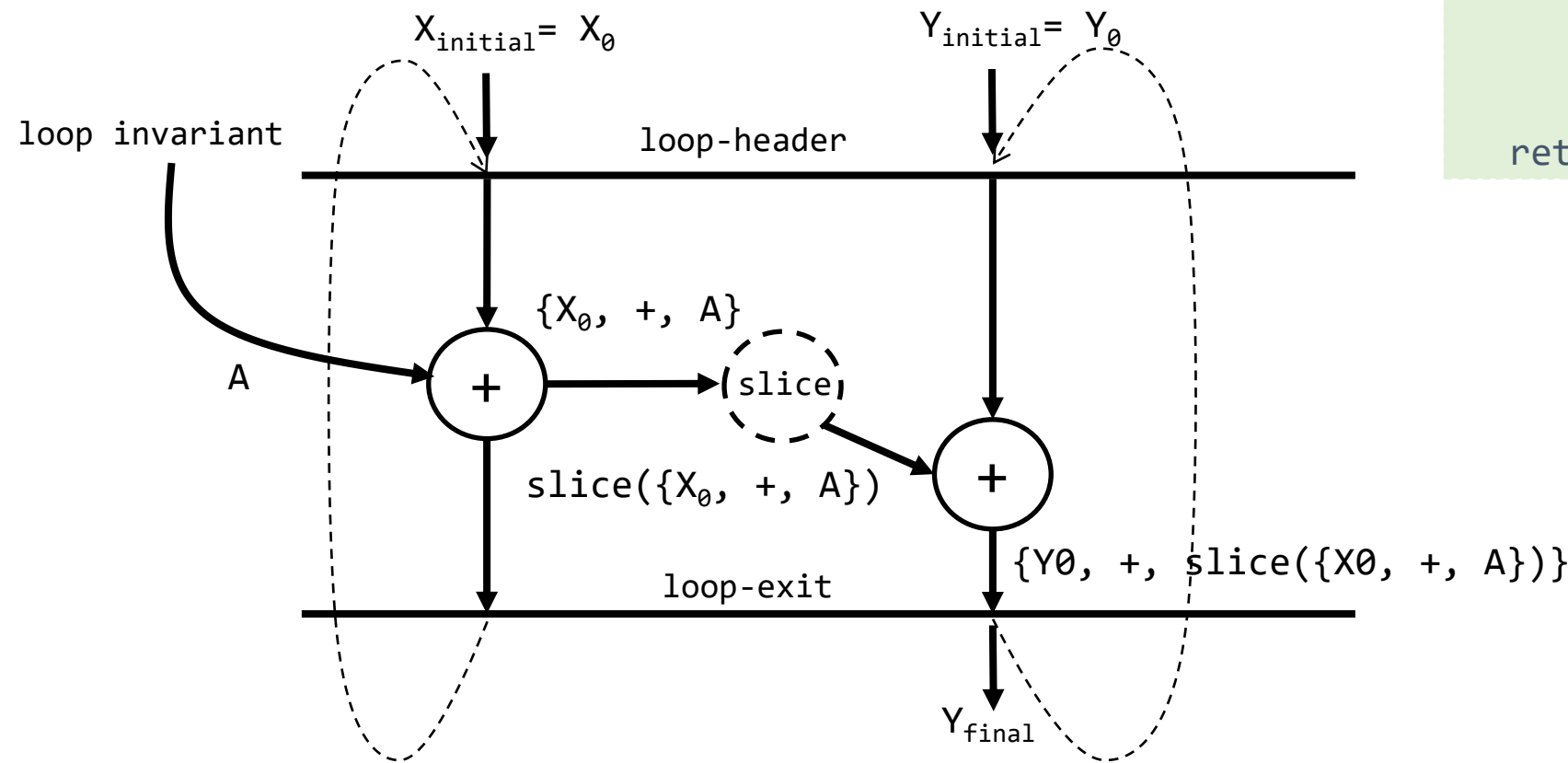


TEV Analysis Pass



# TEV Pass - Opt

```
# PyTorch code.
def forward(self, a, x, y):
    for _ in range(15):
        x = x + a
        ...
        z = x[1,:]
        y = y + z
    return y
```



## Evaluation of $Y_k$

$$Y_k = \{Y_0, +, S(\{X_0, +, A\})\}_k$$

$$\rightarrow Y_k = \{Y_0, +, S(\{X_0, +, A\})\}_k$$

$$\rightarrow Y_k = \{Y_0, +, \{S(X_0), +, S(A)\}\}_k$$

$$\rightarrow Y_k = \{Y_0, +, S(X_0), +, S(A)\}_k$$

$$\rightarrow Y_k = Y_0 + k*S(X_0) + k*(k+1)/2*S(A)$$

# TEV Pass - Opt

## Evaluation of $Y_k$

$$Y_k = \{Y_\theta, +, S(\{X_\theta, +, A\})\}_k$$

$$\rightarrow Y_k = \{Y_\theta, +, S(\{X_\theta, +, A\})\}_k$$

$$\rightarrow Y_k = \{Y_\theta, +, \{S(X_\theta), +, S(A)\}\}_k$$

$$\rightarrow Y_k = \{Y_\theta, +, S(X_\theta), +, S(A)\}_k$$

$$\rightarrow Y_k = Y_\theta + k*S(X_\theta) + k*(k+1)/2*S(A)$$

```
# PyTorch code.  
def forward(self, a, x, y):  
    for _ in range(15):  
        x = x + a  
        ...  
        z = x[1,:]   
        y = y + z  
    return y
```

```
# PyTorch code.  
def forward(self, a, x, y):  
    return y + 15*x[1,:] + 15*(15+1)/2*a[1,:]
```

# Conclusion

- TEV is extension of SCEV to Tensors
- Construction of TEV expressions and rewrite-lemmas
  - Complex optimizations on top of TEV (much like SCEV LSR etc)
- Prototyped in internal-compiler
- Potential opt for MLIR lower CFG dialects
  - Looking forward to collaboration and discussions

# Thank you

**Qualcomm**

Follow us on: [in](#) [twitter](#) [instagram](#) [youtube](#) [facebook](#)

For more information, visit us at:

[qualcomm.com](http://qualcomm.com) & [qualcomm.com/blog](http://qualcomm.com/blog)

Nothing in these materials is an offer to sell any of the components or devices referenced herein.

©2018-2023 Qualcomm Technologies, Inc. and/or its affiliated companies. All Rights Reserved.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Other products and brand names may be trademarks or registered trademarks of their respective owners.

References in this presentation to “Qualcomm” may mean Qualcomm Incorporated, Qualcomm Technologies, Inc., and/or other subsidiaries or business units within the Qualcomm corporate structure, as applicable. Qualcomm Incorporated includes our licensing business, QTL, and the vast majority of our patent portfolio. Qualcomm Technologies, Inc., a subsidiary of Qualcomm Incorporated, operates, along with its subsidiaries, substantially all of our engineering, research and development functions, and substantially all of our products and services businesses, including our QCT semiconductor business.

Snapdragon and Qualcomm branded products are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.