Tensor Evolution

Javed Absar, Principal Engineer*
Samarth Narang, Engineer**
Muthu M. Baskaran, Principal Engineer**

* Qualcomm Technologies International, Ltd.
** Qualcomm Technologies, Inc.

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

Scalar Evolution Analysis and Optimization

- 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 τ_1 over natural number N that produces tensor of same shape as T_c
  - an element-wise operator + associative and commutative
  - τ defined as function τ (i) over N

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

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

Mathematical Formulation

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

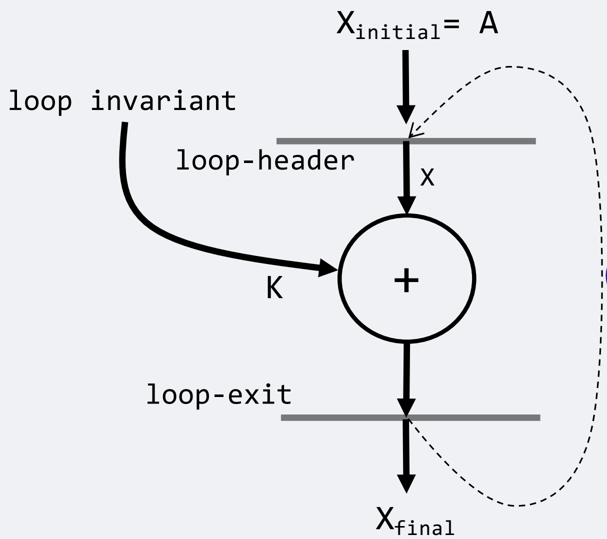
$$\tau = \{Tc_0 , \odot_1, Tc_1 , \odot_2, \ldots, \odot_k, \tau_k\} \qquad \text{eq. 1}$$

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

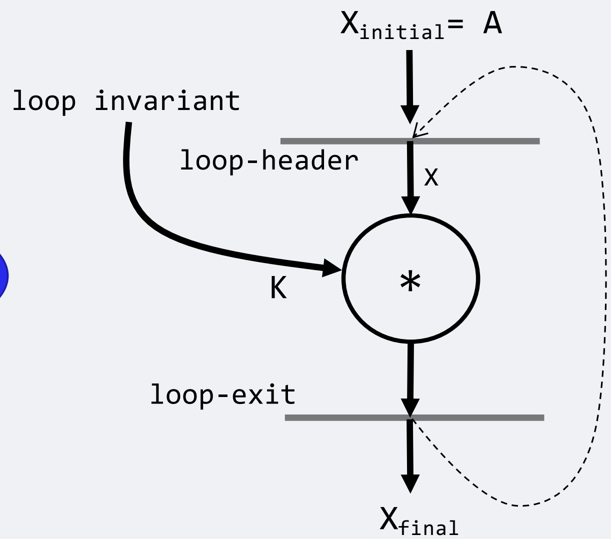
- Note: Operators could be same or different (+,-, *, tanh).

- Recurrences
  - Algebraic properties
  - Computationally reducible at any iteration point

Tensor Evolution – Basic Recurrence



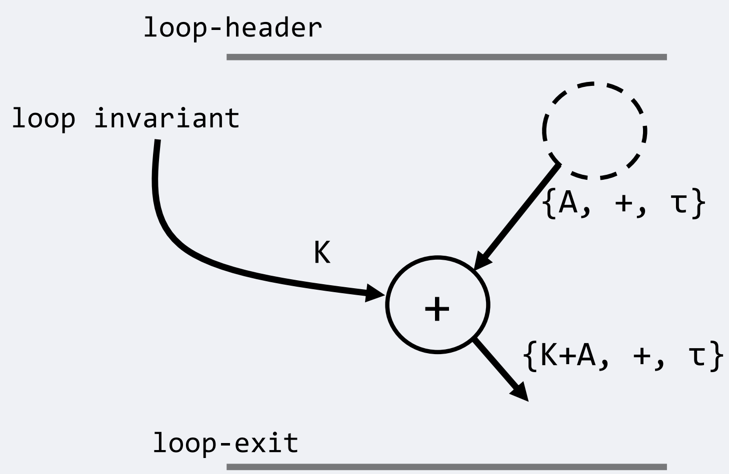
X = {A, +, K}



X = {A, *, K}

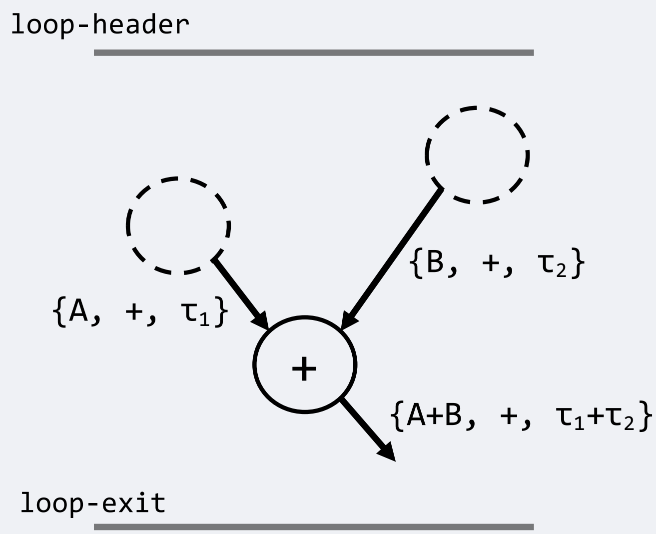
Tensor Evolution

- Lemma: Add a constant (LIV) tensor



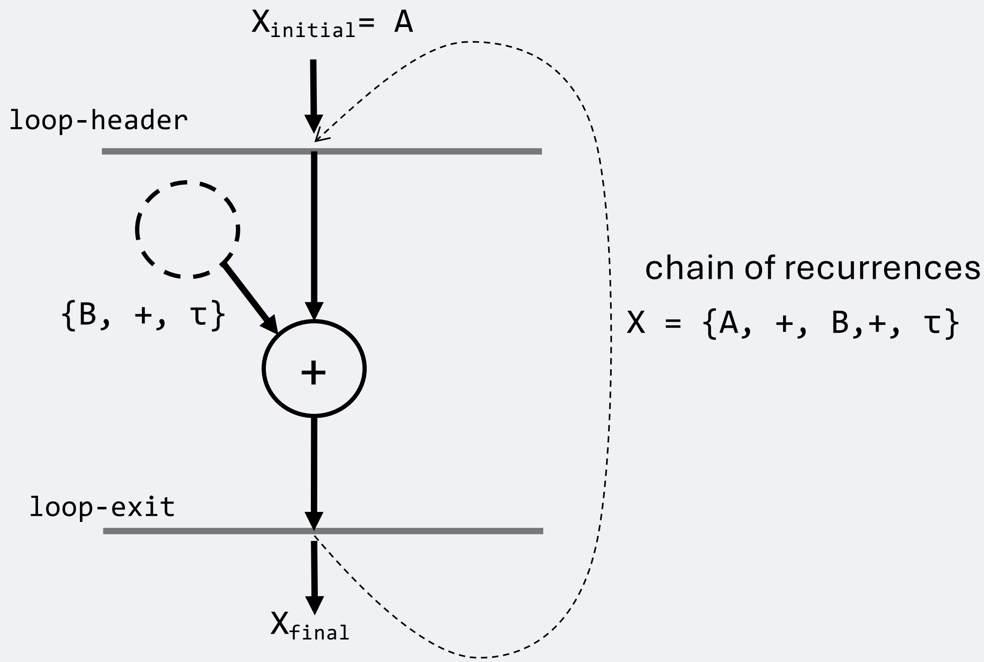
Tensor Evolution

- Lemma: Add two TEVs



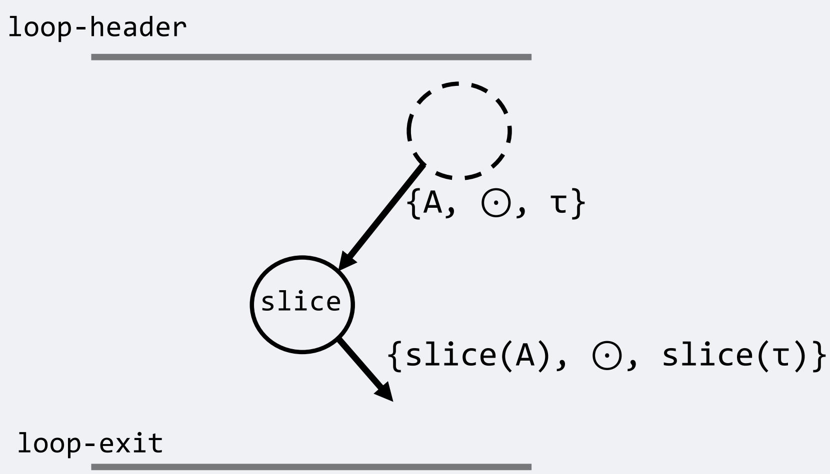
Tensor Evolution

- Lemma: TEV inject into TEV



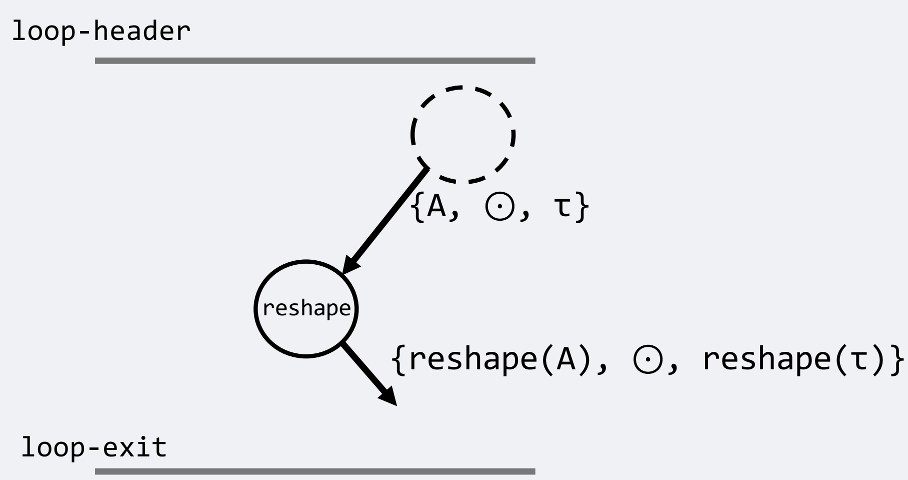
Tensor Evolution

- Lemma: Slice



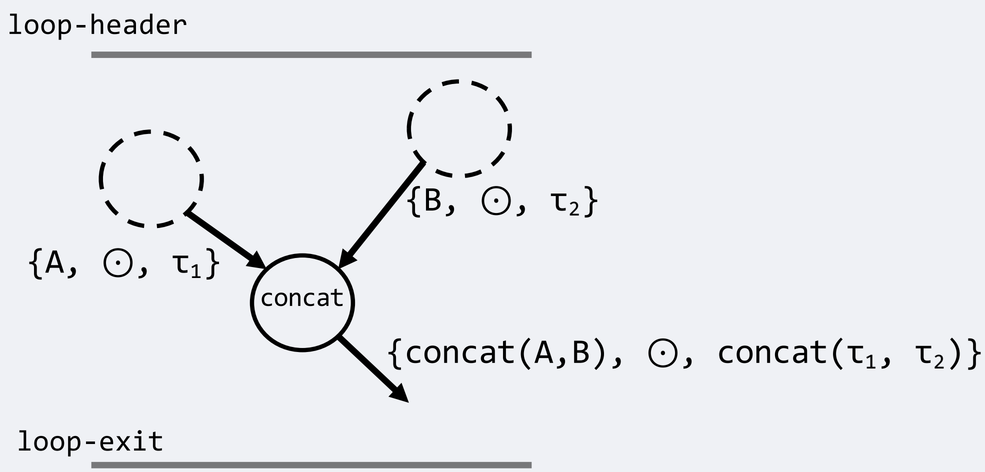
Tensor Evolution

- Lemma: Reshape



Tensor Evolution

- Lemma: Concat



Concat Lemma Proof

- Claim: $C(\{A, \odot, \tau_1\}, \{B, \odot, \tau_2\}) \Rightarrow \{C(A, B), \odot, C(\tau_1, \tau_2)\}$
  - Where C is the concatenation operation, $\odot$ is an element-wise operator.
- Recap: $\{A, \odot, \tau_1\}(i) = A \odot \tau_1(1) \odot \tau_1(2) \odot \ldots \odot \tau_1(i)$ – (1)
  - Where A is a loop-invariant tensor, $\odot$ is an operator, and $\tau_1(i)$ represents evolving tensors.
- Now, consider LHS: $C(\{A, \odot, \tau_1\}(i), \{B, \odot, \tau_2\}(i))$
- $[A \odot \tau_1(1) \odot \ldots \odot \tau_1(i)] \,||\, [B \odot \tau_2(1) \odot \ldots \odot \tau_2(i)]$
  - Where || is the concatenation operator
- Stacking along axis = 0 to preserve structure for simplicity, we get
  - $C(A, B) \odot C(\tau_1(1), \tau_2(1)) \odot \ldots \odot C(\tau_1(i), \tau_2(i))$
  - From (1), the above is equivalent to $\{C(A, B), \odot, C(\tau_1, \tau_2)\}(i)$
- $C(\{A, \odot, \tau_1\}, \{B, \odot, \tau_2\}) = \{C(A, B), \odot, C(\tau_1, \tau_2)\}$

An example for intuition

- Given $A = [1, 2]$, $\tau_1(i) = [i, i]$
- $B = [3, 4]$, $\tau_2(i) = [2i, 2i]$, $\odot = +$
- LHS: $C(\{A, +, \tau_1\}, \{B, +, \tau_2\})(i)$
- $= C([1 + i, 2 + i], [3 + 2i, 4 + 2i])$
- $= [1+i, 2+i, 3+2i, 4+2i]$

- RHS: $\{C(A, B), +, C(\tau_1, \tau_2)\}(i)$
- $= [1, 2, 3, 4] + [i, i, 2i, 2i]$
- $= [1+i, 2+i, 3+2i, 4+2i]$

TeV Injection Lemma Proof

Rewrite Rule:

$\{A, +, \{B, +, \tau\}\} \rightarrow \{A, +, B, +, \tau\}$

The LHS is essentially a CR, examining right-to-left, $\{B, +, \tau\}$ is a BR with B as the loop inv tensor evolving with value τ

1. By definition, a TEV follows:

$\{B, +, \tau\}(i) = B + \tau(1) + \tau(2) + \ldots + \tau(i)$

2. Expanding LHS:

$\{A, +, \{B, +, \tau\}\}(i) = A + \{B, +, \tau\}(1) + \{B, +, \tau\}(2) + \ldots + \{B, +, \tau\}(i)$

$= A + (B + \tau(1)) + (B + \tau(2)) + \ldots + (B + \tau(i))$

$= A + iB + (\tau(1) + \tau(2) + \ldots + \tau(i))$

4. Recognizing the pattern, we get:

$\{A, +, iB, +, \tau\}(i) = A + iB + \tau(1) + \tau(2) + \ldots + \tau(i)$

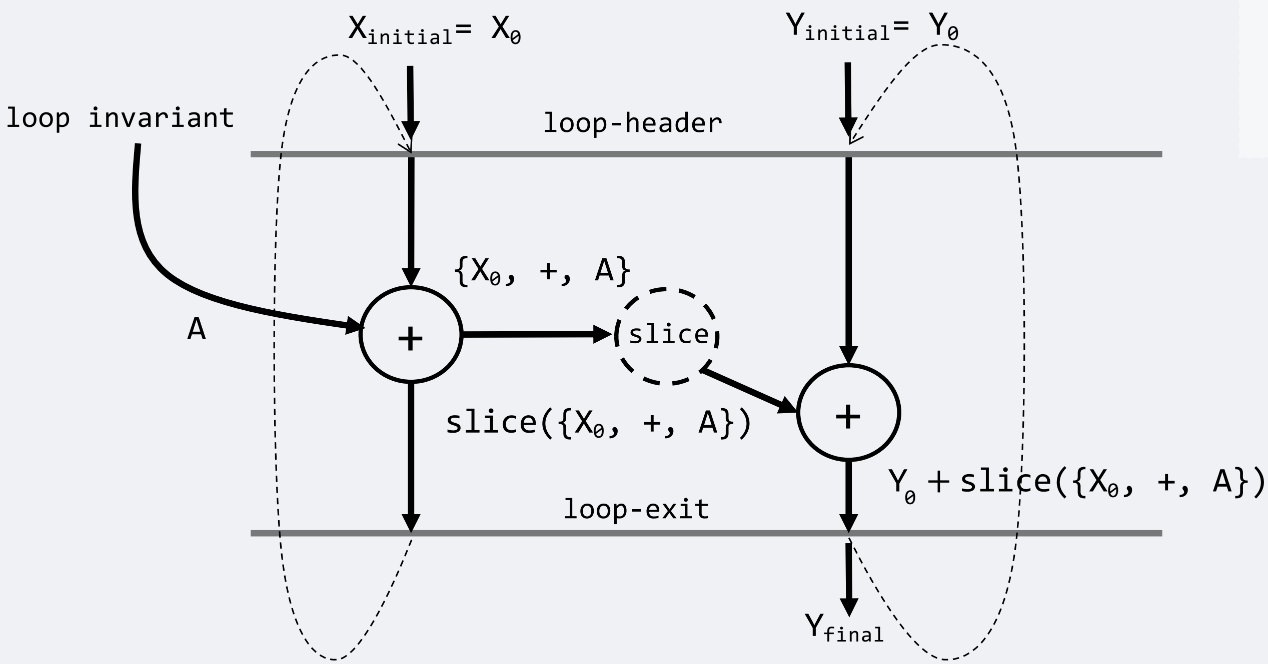
Now, it is easy to see that this is indeed a flattened recurrence as τ changes over loop iterations with A, B being loop invariant tensors.

Tensor Evolution

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

operator	TEV expression	rewrite rule
slice	slice({A, +, τ})	{slice(A), +, slice(τ)}
reshape	reshape({A, $\odot$, τ})	{reshape(A), $\odot$, reshape(τ)}
concat	concat({A, $\odot$, τ_1},{B, $\odot$, τ_2})	{concat(A,B), $\odot$, concat(τ_1, τ_2)}
add K	K + {A, +, τ}	{K+A, +, τ}
add TEVs	{A, +, τ_1} + {B, +, τ_2}	{A+B, +, τ_1+τ_2}
mul	K * {A, +, τ}	{K*A, +, K*τ}
inject TEV	{A, +, {B, +, τ}}	{A, +, B, +, τ}

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

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

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

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

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

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

TEV Pass - Opt

Evaluation of Y_k

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

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

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

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

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

```
# 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

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

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.

Follow us on:

For more information, visit us at qualcomm.com & qualcomm.com/blog