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
Scalar Evolution

- SCEV analysis and opt

```c
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.  \(-\rightarrow \{0,+,%k\}<\%for.body>\) Exits: ((-1 + %n) * %k)
6. ...
Tensor Evolution – Motivating Example 1

• Tensor Evolution Optimization

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

```python
# 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 $N$

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

\[
\{T_c, +, \tau_1\}(i) = T_c + \tau_1(0) + \tau_1(1) + \ldots + \tau_1(i - 1) \quad \text{eq. 2}
\]
Mathematical Formulation

• Chain of Recurrences (Tensor Evolution)
  • loop-invariant tensors $T_{c_0}, T_{c_1}, T_{c_2}, \ldots, T_{c_{i-1}}$;
  • function $\tau_k$ defined over $\mathbb{N}$,
  • operators $\odot_1, \odot_2, \ldots, \odot_k$,
  • chain of evolution of tensor value represented by tuple

  \[
  \tau = \{T_{c_0}, \odot_1, T_{c_1}, \odot_2, \ldots, \odot_k, \tau_k\} \quad \text{eq. 1}
  \]

  \[
  \tau(i) = \{T_{c_0}, \odot_1, \{T_{c_1}, \odot_2, \ldots, \odot_k, \tau_k\}\}(i) \quad \text{eq. 2}
  \]

• 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

<table>
<thead>
<tr>
<th>operator</th>
<th>TEV expression</th>
<th>rewrite rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>slice</td>
<td>slice({A, +, τ})</td>
<td>{slice(A), +, slice(τ)}</td>
</tr>
<tr>
<td></td>
<td>slice({A, *, τ})</td>
<td>{slice(A), *, slice(τ)}</td>
</tr>
<tr>
<td>reshape</td>
<td>reshape({A, ⊙, τ})</td>
<td>{reshape(A), ⊙, reshape(τ)}</td>
</tr>
<tr>
<td>concat</td>
<td>concat({A, ⊙, τ₁}, {B, ⊙, τ₂})</td>
<td>{concat(A, B), ⊙, concat(τ₁, τ₂)}</td>
</tr>
<tr>
<td>add K</td>
<td>K + {A, +, τ}</td>
<td>{K+A, +, τ}</td>
</tr>
<tr>
<td>add TEVs</td>
<td>{A, +, τ₁} + {B, +, τ₂}</td>
<td>{A+B, +, τ₁+τ₂}</td>
</tr>
<tr>
<td>mul</td>
<td>K * {A, +, τ}</td>
<td>{K<em>A, +, K</em>τ}</td>
</tr>
<tr>
<td>inject TEV</td>
<td>{A, +, {B, +, τ}}</td>
<td>{A, +, B, +, τ}</td>
</tr>
</tbody>
</table>
Tensor Evolution – Basic Recurrence

X_{initial} = A

X = \{A, +, K\}

X_{final}

loop invariant

loop-header

K

loop-exit

X_{initial} = A

X = \{A, *, K\}

X_{final}
Tensor Evolution

• Lemma: Add a constant (LIV) tensor
Tensor Evolution

• Lemma: Add two TEVs

\[
\text{loop-header}
\]

\[
\{A, +, \tau_1\} \rightarrow + \rightarrow \{B, +, \tau_2\} \\
\{A+B, +, \tau_1+\tau_2\} \rightarrow \text{loop-exit}
\]
Tensor Evolution

• Lemma: TEV inject into TEV

\[ X_{\text{initial}} = A \]

\[ X_{\text{final}} = \{A, +, B, +, \tau\} \]

chain of recurrences

\[ X = \{A, +, B, +, \tau\} \]
Tensor Evolution

• Lemma: Slice

```
{slice(A), ⊙, slice(τ)}

loop-header

{slic(A), ⊙, slice(τ)}

loop-exit
```
Tensor Evolution

• Lemma: Reshape

\[ \text{loop-header} \]

\[ \{A, \odot, \tau\} \]

\[ \text{reshape} \]

\[ \{\text{reshape}(A), \odot, \text{reshape}(\tau)\} \]

\[ \text{loop-exit} \]
Tensor Evolution

• Lemma: Concat
Tensor Evolution

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

<table>
<thead>
<tr>
<th>operator</th>
<th>TEV expression</th>
<th>rewrite rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>slice</td>
<td>slice({A, +, τ})</td>
<td>{slice(A), +, slice(τ)}</td>
</tr>
<tr>
<td></td>
<td>slice({A, *, τ})</td>
<td>{slice(A), *, slice(τ)}</td>
</tr>
<tr>
<td>reshape</td>
<td>reshape({A, ⊙, τ})</td>
<td>{reshape(A), ⊙, reshape(τ)}</td>
</tr>
<tr>
<td>concat</td>
<td>concat({A, ⊙, τ₁}, {B, ⊙, τ₂})</td>
<td>{concat(A,B), ⊙, concat(τ₁, τ₂)}</td>
</tr>
<tr>
<td>add K</td>
<td>K + {A, +, τ}</td>
<td>{K+A, +, τ}</td>
</tr>
<tr>
<td>add TEVs</td>
<td>{A, +, τ₁} + {B, +, τ₂}</td>
<td>{A+B, +, τ₁+τ₂}</td>
</tr>
<tr>
<td>mul</td>
<td>K * {A, +, τ}</td>
<td>{K<em>A, +, K</em>τ}</td>
</tr>
<tr>
<td>inject TEV</td>
<td>{A, +, {B, +, τ}}</td>
<td>{A, +, B, +, τ}</td>
</tr>
</tbody>
</table>
TEV Pass - Analysis

loop header

loop exit

X\(_{\text{final}}\) Y\(_{\text{final}}\) scan output

TEV Analysis Pass

loop header

loop exit

X\(_{\text{final}}\) Y\(_{\text{final}}\) scan output
Evaluation of $Y_k$

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

- $Y_k = \{Y_0, +, S(\{X_0, +, A\})\}_k$
- $Y_k = \{Y_0, +, S(X_0), +, S(A)\}_k$
- $Y_k = \{Y_0, +, S(X_0), +, S(A)\}_k$
- $Y_k = Y_0 + kS(X_0) + k(k+1)/2S(A)$
TEV Pass - Opt

Evaluation of $Y_k$

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

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

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

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

# 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