CHAIR FOR COMPILER CONSTRUCTION

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

TeIL: a type-safe imperative Tensor Intermediate Language

Norman A. Rink and Jeronimo Castrillon

Technische Universität Dresden, Germany

{norman.rink,jeronimo.castrillon}@tu-dresden.de

ARRAY 2019

22 June 2019

Phoenix, AZ, USA

Tensor Languages

- Recent years have seen an inflation of *tensor frameworks*
  - Each with its own *tensor language*

Frameworks

PYTORCH | Tensor Comprehensions | tvm | TensorFlow

Just a selection of frameworks, apologies if your favorite is not listed

- tensor = (high-dimensional) array

- Functional array languages enjoy type-safety properties
  - E.g. absence of out-of-bounds accesses in well-typed programs (aka. memory-safety)

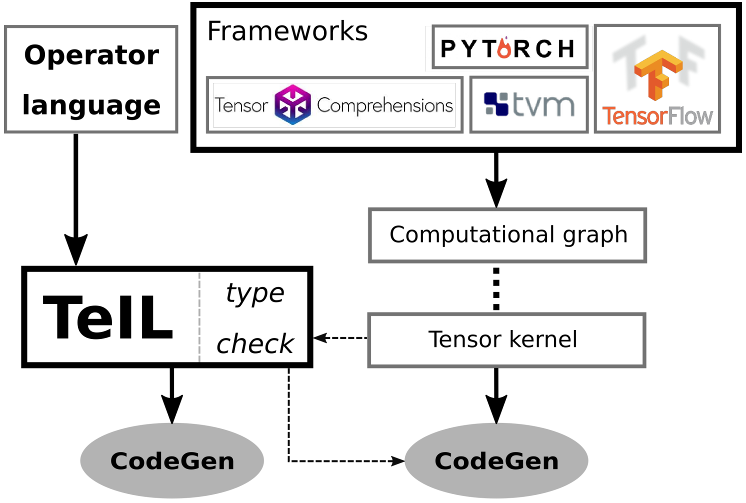
- Recent tensor languages are imperative
  - What about type-safety for imperative tensor/array languages?

Type-Safety, Formal Specification, and TeIL

- Need a formal specification for formal type-safety results
  - Tensor languages from the frameworks not formally specified

- TeIL: an imperative Tensor Intermediate Language
  - Common denominator for reasoning about imperative tensor languages
  - Formal specification and type-safety in Coq
  - No out-of-bounds accesses in well-typed TeIL programs

collective operations, aka. combinators



CHAIR FOR COMPILER CONSTRUCTION

Outline

1. A Motivating Example
2. The TelL Language
3. Core-TelL: Type-Safety
4. Directions for future work

Outline

1. A Motivating Example
2. The TelL Language
3. Core-TelL: Type-Safety
4. Directions for future work

The TVM Framework: Example Kernel

TVM – *Tensor Virtual Machine*

(T Chen, T Moreau Kamil, Z Jiang, L Zheng, E Yan, H Shen, M Cowan, Lwang, Y Hu, L Ceze, C Guestrin, A Krishnamurthy. OSDI 2018)

```
A = placeholder((m,h), name='A')
B = placeholder((n,h), name='B')
k = reduce_axis((0, h), name='k')
C = compute((m, n), lambda i, j:
      sum(A[k, i] * B[k, j], axis=k))
```

$$C_{ij} = \sum_{k=1}^{h} A_{ki} B_{kj}$$

Segmentation fault or silent data corruption. ✗

```
A = placeholder((h,m), name='A')
B = placeholder((h,n), name='B')
k = reduce_axis((0, h), name='k')
C = compute((m, n), lambda i, j:
      sum(A[k, i] * B[k, j], axis=k))
```

$$C_{ij} = \sum_{k=1}^{h} A_{ki} B_{kj}$$

✓

Outline

1. A Motivating Example
2. **The TelL Language**
3. Core-TelL: Type-Safety
4. Directions for future work

CHAIR FOR COMPILER CONSTRUCTION

TelL: Overview

Syntax:

⟨*program*⟩	::=	⟨*alloc*⟩* ⟨*stmt*⟩*
⟨*alloc*⟩	::=	**alloc** ⟨*id*⟩ **: [***i*, …, *i***]**
⟨*stmt*⟩	::=	⟨*id*⟩ **=** ⟨*expr*⟩
⟨*expr*⟩	::=	⟨*id*⟩ | **(** ⟨*expr*⟩ **)**
		| **add** ⟨*expr*⟩ ⟨*expr*⟩ | **mul** ⟨*expr*⟩ ⟨*expr*⟩
		| **prod** ⟨*expr*⟩ ⟨*expr*⟩ | **red+** *i* ⟨*expr*⟩
		| **transp** *i i* ⟨*expr*⟩ | **diag** *i i* ⟨*expr*⟩
		| **expa** *i i* ⟨*expr*⟩ | **proj** *i i* ⟨*expr*⟩

Memory:

$$\mu : \langle id \rangle \rightarrow (\textit{list of}\ \mathrm{Nat}) \rightarrow \mathbb{D}$$

Update of μ at name y and indices $\bar{\kappa}$:

$$\mu'\ x\ \bar{\iota} = \begin{cases} r, & \text{if } x = y \text{ and } \bar{\iota} = \bar{\kappa} \\ \mu\ x\ \bar{\iota}, & \text{otherwise} \end{cases}$$

- Tensor-valued variables are declared with **alloc**
- Declaration assigns a type (shape) to the variable
- Expressions are built from *combinators* (collective operations)

TelL: Expression Typing

Typing context (*shape* assignment):

$$\Gamma : \langle id \rangle \rightarrow (\textit{list of } \mathrm{Nat})$$

Expression typing (subset of rules):

$$\frac{\Gamma(x) = \bar{\iota}}{\Gamma \vdash x : \bar{\iota}} \text{ T-Var} \qquad \frac{\Gamma \vdash e : \bar{\iota}}{\Gamma \vdash (e) : \bar{\iota}} \text{ T-Paren}$$

$$\frac{\Gamma \vdash e_0 : \bar{\iota}_0 \quad \Gamma \vdash e_1 : \bar{\iota}_1}{\Gamma \vdash \texttt{prod}\, e_0\, e_1 : (\bar{\iota}_0 \# \bar{\iota}_1)} \text{ T-Prod}$$

$$\frac{\Gamma \vdash e_0 : \bar{\iota} \quad \Gamma \vdash e_1 : \bar{\iota}}{\Gamma \vdash \texttt{add}\, e_0\, e_1 : \bar{\iota}} \text{ T-Add}$$

$$\frac{\Gamma \vdash e : [n_1, \ldots, n_i, \ldots, n_k]}{\Gamma \vdash \texttt{red}_+\, i\, e : [n_1, \ldots, n_k]} \text{ T-Red}_+$$

$$\frac{\Gamma \vdash e : [n_1, \ldots, n_{i_0}, \ldots, n_{i_1}, \ldots, n_k]}{\Gamma \vdash \texttt{transp}\, i_0\, i_1\, e : [n_1, \ldots, n_{i_1}, \ldots, n_{i_0}, \ldots, n_k]} \text{ T-Transp}$$

$$\frac{\Gamma \vdash e : [n_1, \ldots, n_{i_0}, \ldots, n_{i_1}, \ldots, n_k] \quad n_{i_0} = n_{i_1}}{\Gamma \vdash \texttt{diag}\, i_0\, i_1\, e : [n_1, \ldots, n_{i_0}, \ldots, n_k]} \text{ T-Diag}$$

TeIL: Expression Evaluation

$[\![\langle expr \rangle]\!] : context \to memory \to (list\ of\ \mathrm{Nat}) \to \mathbb{D}$

<u>Example (matrix multiplication):</u>

$$C_{j_1 j_2} = \sum_{k=1}^{h} A_{j_1 k} B_{k j_2}$$

$$\mathbf{alloc}\, A : [l, m]$$
$$\mathbf{alloc}\, B : [m, n]$$
$$\mathbf{alloc}\, C : [l, n]$$
$$C = \mathtt{red}_+\, 2\, (\mathtt{diag}\, 2\, 3\, (\mathtt{prod}\, A\, B))$$

$$\begin{aligned}
&[\![\mathtt{red}_+\, 2\, (\mathtt{diag}\, 2\, 3\, (\mathtt{prod}\, A\, B))]\!]\, \Gamma\, \mu\, [j_1, j_2] \\
&\quad = \sum_k [\![\mathtt{diag}\, 2\, 3\, (\mathtt{prod}\, A\, B)]\!]\, \Gamma\, \mu\, [j_1, k, j_2] \\
&\quad = \sum_k [\![\mathtt{prod}\, A\, B]\!]\, \Gamma\, \mu\, [j_1, k, k, j_2] \\
&\quad = \sum_k ([\![A]\!]\, \Gamma\, \mu\, [j_1, k]) \cdot ([\![B]\!]\, \Gamma\, \mu\, [k, j_2]) \\
&\quad = \sum_k (\mu\, A\, [j_1, k]) \cdot (\mu\, B\, [k, j_2])
\end{aligned}$$

TeIL: Program Evaluation

Program evaluation:

$$\frac{\Gamma(x) = \bar{\iota} \quad \forall \bar{\kappa} \leq \bar{\iota}.\ \bar{\kappa} \in dom(\mu\, x) \qquad \forall \bar{\kappa} \leq \bar{\iota}.\ \text{let } r_{\bar{\kappa}} = [\![e]\!]\,\Gamma\,\mu\,\bar{\kappa}}{\langle \mu,\ x = e \rangle \longrightarrow_{\Gamma} \mu\{x \mapsto \lambda\bar{\kappa}.r_{\bar{\kappa}}\}}\ \text{St-Stmt}$$

$$\frac{}{\langle \mu,\ \ \rangle \longrightarrow_{\Gamma} \mu}\ \text{St-Empty}$$

$$\frac{\langle \mu',\ stmts \rangle \longrightarrow_{\Gamma} \mu' \quad \langle \mu',\ stmt \rangle \longrightarrow_{\Gamma} \mu''}{\langle \mu,\ stmts\ stmt \rangle \longrightarrow_{\Gamma} \mu''}\ \text{St-Seq}$$

$$\frac{\langle \mu_{allocs},\ stmts \rangle \longrightarrow_{\Gamma_{allocs}} \mu'}{\langle \mu_{allocs},\ allocs\ stmts \rangle \Downarrow \mu'}\ \text{Eval-Prog}$$

Program typing:

$$\frac{\Gamma(x) = \bar{\iota} \qquad \Gamma \vdash e : \bar{\iota}}{\Gamma \vdash x = e : \texttt{ok}}\ \text{OK-Stmt}$$

Evaluation can only proceed if there are **no out-of-bounds accesses.**

$$\frac{\Gamma \vdash stmts : \texttt{ok} \qquad \Gamma \vdash stmt : \texttt{ok}}{\Gamma \vdash stmts\ stmt : \texttt{ok}}\ \text{OK-Seq}$$

$$\frac{\Gamma_{allocs} \vdash stmts : \texttt{ok}}{\Gamma_{allocs} \vdash allocs\ stmts : \texttt{ok}}\ \text{OK-Prog}$$

Type-safety:
well-typed programs can be fully evaluated

Outline

1. A Motivating Example
2. The TelL Language
3. Core-TelL: Formal Development of Type-Safety
4. Directions for future work

CHAIR FOR COMPILER CONSTRUCTION

Core-TeIL: Overview

- Types/shapes and index lists are modelled as lists of natural numbers
  - Need a number of straightforward results about list operations

- Formal reasoning simplifies if we restrict the manipulation of types/shapes
  - E.g. let transpositions act only on adjacent dimensions

No loss of generality by a standard result of group theory.

$$\frac{\Gamma \vdash e : [n_1, \ldots, n_i, n_{i+1}, \ldots, n_k]}{\Gamma \vdash \texttt{transp}\, i\; e : [n_1, \ldots, n_{i+1}, n_i, \ldots, n_k]} \text{ T-Transp}^{core}$$

$$\frac{\Gamma \vdash e : [n_1, n_2, \ldots, n_k]}{\Gamma \vdash \texttt{red}_+\, e : [n_2, \ldots, n_k]} \text{ T-Red}_+^{core}$$

$$\frac{\Gamma \vdash e : [n_1, n_2, \ldots, \ldots, n_k] \quad n_1 = n_2}{\Gamma \vdash \texttt{diag}\, e : [n_2, \ldots, n_k]} \text{ T-Diag}^{core}$$

Core-TeIL: Type-Safety

Theorem (type-safety):

If $\Gamma_{allocs} \vdash allocs\ stmts : \mathrm{ok}$, then there exists a memory μ' such that

- $\langle \mu_{allocs}, allocs\ stmts \rangle \Downarrow \mu'$
- $\mu' \sim \mu_{allocs}$.

(Theorem 4.6)

Lemma:

If $\Gamma_{allocs} \vdash x = e : \mathrm{ok}$ and $\mu \sim \mu_{allocs}$, then there exists a memory μ' such that

- $\langle \mu_{allocs}, x = e \rangle \rightarrow \mu'$
- $\mu' \sim \mu_{allocs}$.

(Lemma 4.5)

<u>Equivalence of memories:</u>

$\mu_1 \sim \mu_2$ iff the memories μ_1 and μ_2 have the same domains.

<u>Type-safety (aka. memory safety) for reads:</u>

Lemma:

If $\Gamma_{allocs} \vdash e : \bar{\iota}$ and $\mu \sim \mu_{allocs}$, then $[\![e]\!]\, \Gamma_{allocs}\, \mu\, \bar{\kappa}$ is well-defined for all $\bar{\kappa} \leq \bar{\iota}$.

(Lemma 4.3)

https://github.com/normanrink/TensorIR

Outline

1. A Motivating Example
2. The TelL Language
3. Core-TelL: Formal Development of Type-Safety
4. Directions for future work

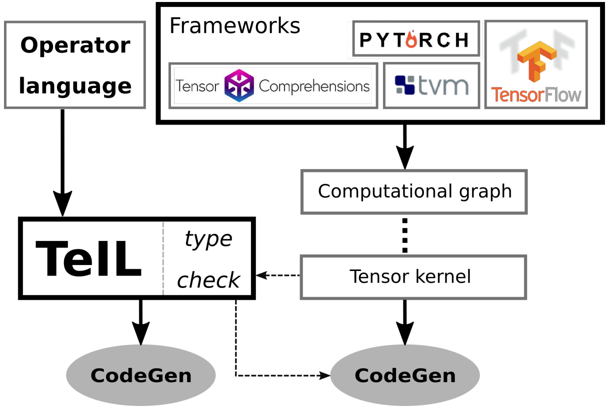
CHAIR FOR COMPILER CONSTRUCTION

Directions for future work

- Application: language for tensor operators in CFD
  https://github.com/normanrink/cfdlang/tree/operators
  - Use TeIL as a typed intermediate language
  - Equational reasoning for validating transformations
- Stencil kernels cannot currently be expressed in TeIL
  - Extend TeIL analogously to recent extensions of Lift

  (B Hagedorn, L Stoltzfus, M Steuwer, S Gorlatch, C Dubach. CGO 2018)
- Variation/instanciation of the abstract memory model
  - Potential application to performance portability between array languages

  (A Šinkarovs, R Bernecky, H-N Vießmann, S-B Scholz. ARRAY 2018)
- Reasoning about data races in parallel execution of TeIL



TeIL: a type-safe imperative Tensor Intermediate Language

Norman A. Rink and Jeronimo Castrillon
Technische Universität Dresden, Germany
{norman.rink,jeronimo.castrillon}@tu-dresden.de

ARRAY 2019
22 June 2019
Phoenix, AZ, USA

Thank you