3 1/2 Questions for Code Generation and Compilation of Tensor Languages

Norman A. Rink
Technische Universität Dresden, Germany
norman.rink@tu-dresden.de

22 January 2020
Recent years have seen an inflation of tensor frameworks. Each with its own tensor language.

Tensor = (high-dimensional) array

Tensor/array languages consider arrays as primitive units of data – not (just scalars).

Operations applied to entire arrays at once (collective operations)

“Classical” examples of array languages:
- APL, Fortran, Matlab, R, C++/Eigen, Python/Numpy
- SaC, ML and Haskell with array libraries (e.g. Repa, Accelerate), Futhark, Lift

Conveniences/abstractions as found in tensor frameworks may not be offered!
Outline

1. Thread-parallelism for multi-core CPUs
2. (Memory) Safety and type/shape discipline
3. Towards provably correct compilation
4. How much should we care?
5. Summary

3 questions/challenges

1/2 a question/challenge
Outline

1. Thread-parallelism for multi-core CPUs
2. (Memory) Safety and type/shape discipline
3. Towards provably correct compilation
4. How much should we care?
5. Summary
Tensor kernels on multi-core CPUs

Performance on a set of machine learning and fluid dynamics kernels:

E.g. mttkrp kernel:

```c
for (int i = 0; i <= (I-1); i++)
  for (int j = 0; j <= (J-1); j++)
    for (int k = 0; k <= (K-1); k++)
      for (int l = 0; l <= (L-1); l++)
        A[i][j] = B[i][k][l] * D[l][j] * C[k][j];
```

CNN on multi-core CPUs

CNN for recognition of digits 0…9 from 28x28 images (MNIST data set):


Why does multi-core performance not scale (in Tensorflow or PyTorch)?

SaC: “Single Assignment C”, array language

Multi-core CPUs performance and scaling:

Outline

1. Thread-parallelism for multi-core CPUs
2. (Memory) Safety and type/shape discipline
3. Towards provably correct compilation
4. How much should we care?
5. Summary
**The TVM Framework: Example Kernel**

**TVM – Tensor Virtual Machine**

\[
A = \text{placeholder}((m, h), \text{name}='A')
\]

\[
B = \text{placeholder}((n, h), \text{name}='B')
\]

\[
k = \text{reduce_axis}((0, h), \text{name}='k')
\]

\[
C = \text{compute}((m, n), \text{lambda} \ i, j:
\sum A[k, i] \ast B[k, j], \text{axis} = k)
\]

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

Segmentation fault or silent data corruption.

\[
C_{ij} = \sum_{k=1}^{h} A_{ki} B_{kj}
\]
TelL: Overview

- TelL: 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 TelL programs

Syntax:

\[
\begin{align*}
\text{program} & \ ::= \ (\text{alloc})^{*} \ (\text{stmt})^{*} \\
\text{alloc} & \ ::= \ \text{alloc} \ \langle \text{id} \rangle : [i, \ldots, i] \\
\text{stmt} & \ ::= \ \langle \text{id} \rangle = \ (\text{expr}) \\
\text{expr} & \ ::= \ \langle \text{id} \rangle \ | \ (\text{expr}) \ | \ \text{add} \ (\text{expr}) \ (\text{expr}) \ | \ \text{mul} \ (\text{expr}) \ (\text{expr}) \\
& \quad \ | \ \text{prod} \ (\text{expr}) \ (\text{expr}) \ | \ \text{red+} \ i \ (\text{expr}) \ | \ \text{transp} \ i \ i \ (\text{expr}) \\
& \quad \ | \ \text{diag} \ i \ i \ (\text{expr}) \ | \ \text{expa} \ i \ i \ (\text{expr}) \ | \ \text{proj} \ i \ i \ (\text{expr})
\end{align*}
\]

- Declarations assign types/shapes to variables
- Expressions are built from collective operations
Tell: Type-Safety

Theorem (type-safety):
If $\Gamma_{allocs} \vdash allocs \ stmts : ok$, then there exists a memory $\mu'$ such that
- $\langle \mu_{allocs}, allocs \ stmts \rangle \downarrow \mu'$
- $\mu' \sim \mu_{allocs}$

Type-safety (aka. memory safety) for reads:

Lemma:
If $\Gamma_{allocs} \vdash e : \bar{t}$ and $\mu \sim \mu_{allocs}$, then $\llbracket e \rrbracket_{\Gamma_{allocs}} \mu \bar{k}$ is well-defined for all $\bar{k} \leq \bar{t}$.

Equivalence of memories:
$\mu_1 \sim \mu_2$ iff the memories $\mu_1$ and $\mu_2$ have the same domains.

Type-safety (aka. memory safety) for writes:

Lemma:
If $\Gamma_{allocs} \vdash x = e : ok$ and $\mu \sim \mu_{allocs}$, then there exists a memory $\mu'$ such that
- $\langle \mu_{allocs}, x = e \rangle \rightarrow \mu'$
- $\mu' \sim \mu_{allocs}$.

https://github.com/normanrink/TensorIR

Is Tell too strict?

- **Tell** enforces a strict discipline for the types/shapes of tensors:
  - Every variable has to be declared with a specific shape
  - Operations will only work if argument shapes match exactly

- **Contrast with Numpy/Tensorflow:**
  - Leads to interesting effects during learning
    - One-hot encoding for MNIST labels:

```python
x = np.array(range(4))  # x.shape = (4,)
z = np.eye(4)           # z.shape = (4, 4)
x + z                   # (x + z).shape = (4, 4)

y_train_HOT = np.array([np.eye(1, M=10, k=y) for y in y_train])  # y_train_HOT.shape = (60000, 1, 10)
# should be: (60000, 10)

# This works without error messages:
np.sum((y_pred - y_train_HOT)**2)
```

- **1's prepended to the shape of x**
- **Broadcasting of the resulting first dimension of x**
Outline

1. Thread-parallelism for multi-core CPUs
2. (Memory) Safety and type/shape discipline
3. Towards provably correct compilation
4. How much should we care?
5. Summary
The TACO Framework: Example Kernels

**TACO – Tensor Algebra Compiler**
(F Kjolstad, S Kamil, S Chou, D Lugato, S Amarasinghe. OOPSLA 2017)

\[
A(i,j) = B(i,j,k) \times C(k)
\]

\[
A_{ij} = \sum_{k=1}^{n} B_{ijk} C_k
\]

\[
A(i,j) = B(i,j,k) \times C(i)
\]

\[
A_{ij} = (\sum_{k=1}^{n} B_{ijk}) C_i
\]

\[
A(i,j) = B(k,j,i) \times C(i)
\]

\[
A_{ij} = (\sum_{k=1}^{n} B_{kji}) C_i
\]

Compiler loops on this.
Why provably correct compilation?

- Reliable or trustworthy systems require correct compilation:
  - Compiler bugs can change the meaning of compiled programs
  - Compromises safety and security

- What are correctness requirements for tensor kernels or machine learning models?
  - Possible attack vectors when executed with concurrent processes (e.g. on multi-core CPUs)

- Program verification at source code level relies on correct compilation

https://deepspec.org/main
What is correct compilation?

- **Semantics preservation:**
  - Source program $S$, compiler (function) $\text{comp} : \text{source} \rightarrow \text{target}$, behavior $B$.
  - Then, desire correctness properties along the lines of:
    $$\forall S, B : S \text{ valid } \Rightarrow S \text{ has behavior } B \Rightarrow \text{comp}(S) \text{ has behavior } B$$

- Variations of semantics preservation and other notions of compiler correctness exits.
  - The above definition of semantics preservation looks at single programs in isolation.

- For provable correctness, generally need (at least) the following ingredients:
  - Formal specifications of source and target languages
    - Including definitions of allowed behaviors
  - Definition of $\text{comp} : \text{source} \rightarrow \text{target}$

    E.g. C/C++, Java, ML, Haskell.
    But also DSLs like SQL or TeIL.

Typically machine language.
But also, e.g., LLVM or C as intermediate languages.

Array languages, e.g., Futhark or Lift.
For a correct DSL/TellL compiler, could build on existing provably correct compilers

- CakeML (R Kumar, MO Myreen, M Norrish, S Owens. CakeML: A Verified Implementation of ML. POPL 2014)

In 2011, the Csmith tool found 202 bugs in LLVM, 79 bugs in GCC, 0 bugs in CompCert (middle-end).
(X Yang, Y Chen, E Eide, J Regehr. PLDI 2011)

How realistic is this? Is it worth the effort?
Performance of CompCert and ML

Matrix multiplication, 10 iterations, runtime in seconds:

- Need a more convincing approach to correct compilation for tensor/array languages!

Approximately 10x faster with CompCert compared to other tools.
Outline

1. Thread-parallelism for multi-core CPUs
2. (Memory) Safety and type/shape discipline
3. Towards provably correct compilation
4. How much should we care?
5. Summary
Safety & Correctness: How much should we care?

- Type safety for productivity in the tensor/machine learning domain
  - Usefulness of detection of simple type or shape errors
  - What other errors or productivity issue can be caught by static analysis in tensor languages?

- Memory safety for safety & security:
  - Erroneous predictions in safety-critical machine learning applications (automotive)
  - How can buffer overflows in machine learning applications be exploited?

- Systems, compilers, and PL communities are increasingly presenting formally verified code as part of their research results.

This half-question is the most open-ended one! 
\[ \frac{1}{2} \gg 3 \]
Outline

1. Thread-parallelism for multi-core CPUs
2. (Memory) Safety and type/shape discipline
3. Towards provably correct compilation
4. How much should we care?
5. Summary
The code generation of existing machine learning frameworks for multi-core CPUs can be improved.

- What are good motivating use cases for this? (E.g. inference on edge devices?)

Memory safety – safety & security in general:

- How critical is this for tensor languages or machine learning frameworks and applications?
- Will we see attacks or exploits through machine learning applications? (E.g. running in a shared-memory environment?)

Reliability and trustworthiness:

- Correctness of compilers …
- … generalizes to systems (≈ frameworks), including libraries and runtime support.
- Related to the previous point: bugs in system software could be exploited by attackers.
3 ¹⁄₂ Questions for Code Generation and Compilation of Tensor Languages

Norman A. Rink
Technische Universität Dresden, Germany
norman.rink@tu-dresden.de

22 January 2020