Meta-programming for Cross-Domain Tensor Optimizations

Adilla Susungi, Norman A. Rink, Jeronimo Castrillon, Claude Tadonki

Meta-programming for Cross-Domain Tensor Optimizations

Adilla Susungi, Norman A. Rink, Albert Cohen, Jerónimo Castrillón, Claude Tadonki

adilla.susungi@mines-paristech.fr — norman.rink@tu-dresden.de

Design and semantics of a tensor optimization meta-language

- Functional transformation meta-language with high-level abstractions for tensor computations to enable the composition of different type of transformations (e.g. loop, algebraic or layout transformations).
- Formal specification of tensor operations and loop transformations using denotational semantics.
- Examples of use: empirical tuning engines, meta-programming optimizations by experts.

TeML Grammar

⟨*program*⟩ ::= ⟨*stmt*⟩ ⟨*program*⟩
| ϵ

⟨*stmt*⟩ ::= ⟨*id*⟩ = ⟨*expression*⟩
| ⟨*id*⟩ = @⟨*id*⟩ : ⟨*expression*⟩
| `codegen` (⟨*ids*⟩)
| `init` (...)

⟨*expression*⟩ ::= ⟨*Texpression*⟩
| ⟨*Lexpression*⟩

⟨*Texpression*⟩ ::= `tensor` ([⟨*ints*⟩])
| `eq` (⟨*id*⟩, ⟨*iters*⟩? → ⟨*iters*⟩)
| `vop` (⟨*id*⟩, ⟨*id*⟩, [⟨*iters*⟩?, ⟨*iters*⟩?])
| `op` (⟨*id*⟩, ⟨*id*⟩, [⟨*iters*⟩?, ⟨*iters*⟩?] → ⟨*iters*⟩)

⟨*Lexpression*⟩ ::= `build` (⟨*id*⟩)
| `stripmine` (⟨*id*⟩, ⟨*int*⟩, ⟨*int*⟩)
| `interchange` (⟨*id*⟩, ⟨*int*⟩, ⟨*int*⟩)
| ...

⟨*iters*⟩ ::= [⟨*ids*⟩]

⟨*ids*⟩ ::= ⟨*id*⟩ (, ⟨*id*⟩)*

⟨*ints*⟩ ::= ⟨*int*⟩ (, ⟨*int*⟩)*

Code Sample

```
# -- Begin program specification
w = tensor(double, [13])
u = tensor(double, [13, 13, 13])
L = tensor(double, [13, 13])
M_ = outerproduct([w, w, w])
Lh = div(L, w, [[i1, i2], [i2]] -> [i1, i2])
M  = entrywise_mul(M_, u)
r1 = contract(Lh, M, [[2, 1]])
r2 = contract(Lh, M, [[2, 2]])
r3 = contract(Lh, M, [[2, 3]])
# -- End program specification
# Code generation without transformations
l1 = build(M_)
l2 = build(Lh)
l3 = build(M)
l4 = build(r1)
l5 = build(r2)
l6 = build(r3)
codegen([l1, l2, l3, l4, l5, l6])
# Code generation with loop fusions only
l7 = fuse(l4, l5, 3)
l8 = fuse(l7, l6, 3)
codegen([l1, l2, l3, l7])
# Code generation with fusion, parallelism and vectorization
l9 = parallelize(l1, 1, None)
l10 = parallelize(l2, 1, None)
l11 = parallelize(l3, 1, None)
l12 = parallelize(l8, 1, None)
l13 = vectorize(l9, 3)
l14 = vectorize(l10, 2)
l15 = vectorize(l11, 3)
codegen([l13, l14, l15, l12])
```

Semantics Foundations

Domains and state

$$\mathbf{T} = \{\, \langle (op, S, I), ts\rangle \mid (ts = []) \vee (ts = [t_1, \ldots, t_k] \wedge t_i \in \mathbf{T}) \}$$
$$\mathbf{L} = \{\, \langle id, [x_1, \ldots, x_k]\rangle \mid x_i \in \mathbf{L} \cup \mathbf{T} \,\}$$
$$\sigma : identifier \rightarrow (\mathbf{T} + \mathbf{L})$$

Valuation functions

$$\mathcal{P}_{prog}[\![s\ p]\!] = \mathcal{P}_{prog}[\![p]\!] \circ \mathcal{P}_{stmt}[\![s]\!]$$

$$\mathcal{E}_t[\![\texttt{tensor}(S)]\!] = \lambda\sigma.\langle(\square, S, \epsilon), []\rangle$$

$$\mathcal{E}_t[\![\texttt{eq}(t, I_0 \rightarrow I_1)]\!] =$$
$$\lambda\sigma.\text{let } \langle(op, S, I'), ys\rangle = \sigma(t)$$
$$y = \langle(op, S, I''), ys\rangle$$
$$x = \langle(\square, S', I_1), []\rangle$$
$$\text{in } \langle(=, \bullet, \bullet), [x, y]\rangle\,,$$
$$\text{where } \begin{cases} I' \neq \epsilon \wedge I'' = I'\,, \text{ if } I_0 = \epsilon \\ I' = \epsilon \wedge I'' = I_0\,, \text{ if } I_0 \neq \epsilon \end{cases}$$

$$\mathcal{E}_l[\![\texttt{build}(t)]\!] = \lambda\sigma.\text{let } r = \text{“number of iterators in } \sigma(t)\text{”}$$
$$i_k = (0, ub_k, 1) \text{ for } k = 1, \ldots, r$$
$$\text{in } \langle i_1, \ldots, \langle i_r, [\sigma(t)]\rangle \ldots\rangle\,,$$
$$\text{where } \sigma(t) = \langle(=, \bullet, \bullet), [x, y]\rangle$$

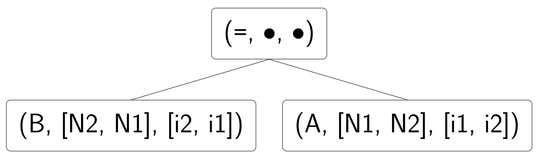
$$\mathcal{E}_l[\![\texttt{stripmine}(l, r, v)]\!] =$$
$$\lambda\sigma.\text{let } \langle i_1, \ldots \langle i_r, xs\rangle \ldots\rangle = \sigma(l)$$
$$(b, e, 1) = i_r$$
$$i'_r = (0, (e - b)/v - 1, 1)$$
$$i'_{r+1} = (b + v \cdot i'_r, b + v \cdot i'_r + (v - 1), 1)$$
$$\text{in } \langle i_1, \ldots \langle i'_r, [\langle i'_{r+1}, xs\rangle]\rangle \ldots\rangle$$

$$\mathcal{E}_l[\![\texttt{interchange}(l, r_1, r_2)]\!] =$$
$$\lambda\sigma.\text{let } \langle i_1, \ldots \langle i_{r_1}, \ldots \langle i_{r_2}, xs\rangle \ldots\rangle \ldots\rangle = \sigma(l)$$
$$\text{in } \langle i_1, \ldots \langle i_{r_2}, \ldots \langle i_{r_1}, xs\rangle \ldots\rangle \ldots\rangle$$

Tensor Expressions

Matrix transposition

```
A = tensor([N1, N2])
B = eq(A, [i1, i2] -> [i2, i1])
```



$$\frac{\mathcal{E}_l[\![\texttt{build}(B)]\!]\sigma_2 = \langle \texttt{i1}, [\langle \texttt{i2}, [\sigma_2(B)]\rangle]\rangle :}{}$$

```
for (int i1 = 0; i1 <= (N1-1); i1++)
 for (int i2 = 0; i2 <= (N2-1); i2++)
  B[i2][i1] = A[i1][i2];
```

Compositions

Contraction

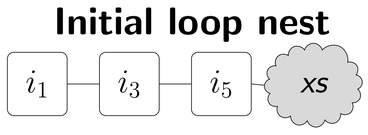
$$\mathcal{P}_{stmt}[\![t' = \texttt{contract}(t_0, t_1, [r_0, r_1])]\!] = \mathcal{P}_{prog}\left[\!\!\left[\begin{array}{l} t_2 = \texttt{vmul}(t_0, t_1, [I, J]) \\ t' = \texttt{add}(t', t_2, [I', \epsilon] \rightarrow I') \end{array} \right]\!\!\right]$$

where

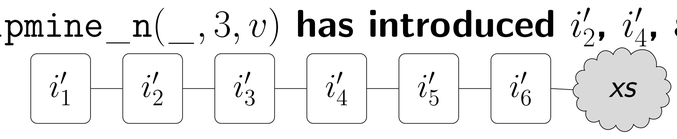
$$I = [\texttt{i0}, \ldots, \texttt{i}(r_0 - 1), \texttt{k}, \texttt{i}(r_0 + 1), \ldots, \texttt{i}s_0]$$
$$J = [\texttt{j0}, \ldots, \texttt{j}(r_1 - 1), \texttt{k}, \texttt{j}(r_1 + 1), \ldots, \texttt{j}s_1]$$
$$I' = (I \backslash \{\texttt{k}\})\ ||\ (J \backslash \{\texttt{k}\})$$

Tiling

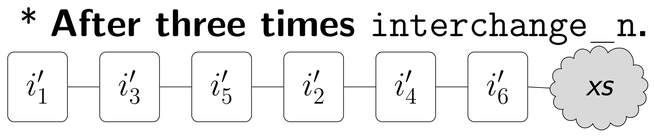
Initial loop nest



`stripmine_n`$(\_, 3, v)$ **has introduced** i'_2**,** i'_4**, and** i'_6

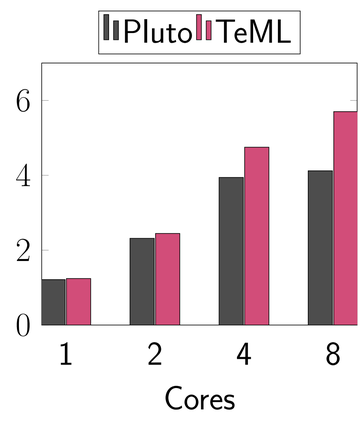


*** After three times** `interchange_n`**.**

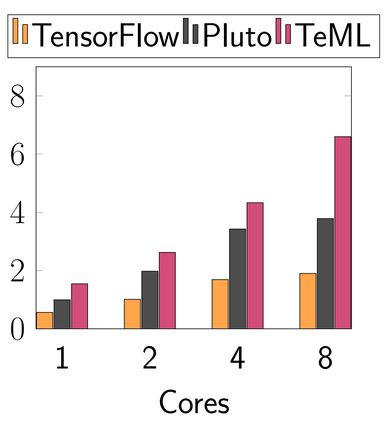


Experiments

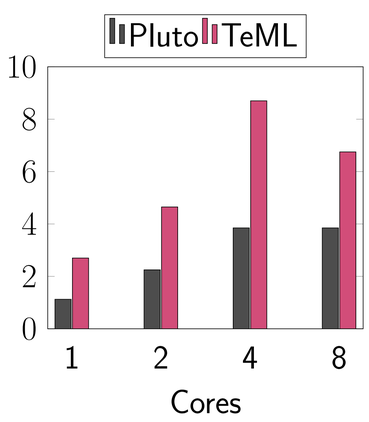
Helmholtz



Mttkrp



Grouped convolutions



- On Intel(R) Core(TM) i7-4910MQ CPU (2.90GHz, 8 hyperthreads, 8192KB of shared L3 cache), Ubuntu 16.04.
- Generated C programs compiled with the Intel C compiler ICC 18.02 (flags: -O3 -xHost -qopenmp)

With TeML:

- Pluto (v 0.11.4) optimizations reproducible
- Capability to express better optimization paths

Future Work

- Abstractions for memory virtualization, stencil patterns, sparse tensors and corresponding semantics, extensions for parallelism support, type system

This work was partially supported by the German Research Council (DFG) through the Cluster of Excellence ‘Center for Advancing Electronics Dresden’ (cfaed) and by PSL University through the ACOPAL project.