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 types 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.

Semantics Foundations

Domains and state

\[ T = \{ (\langle op, S, I \rangle, ts) \mid (ts = []) \lor (ts = \{ t_1, \ldots, t_k \} \land t_i \in T) \} \]

\[ L = \{ (id, \{ x_1, \ldots, x_k \}) \mid x_i \in L \lor T \} \]

\[ \sigma: id \mapsto (T \lor L) \]

Valuation functions

\[ \mathcal{P}_{prog}[P] = \mathcal{P}_{prog}[P] \circ \mathcal{P}_{stmt} \]

\[ E_{\langle \text{tensor} \rangle}(S) = \lambda \sigma. \langle \Box, \sigma, e, [], [] \rangle \]

\[ E_{\langle \text{eq} \rangle}(t_0 \rightarrow t_1) = \lambda \sigma. \lambda r. \langle \Box, \sigma, e, [r], [] \rangle \]

\[ E_{\langle \text{build} \rangle}(t) = \lambda \sigma. \lambda r. \text{let } r = \text{number of iterators in } \sigma \]

\[ E_{\langle \text{stripmine} \rangle}(l, r, v) = \lambda \sigma. \lambda r. \text{let } \{ \{ \ldots \} \} = \sigma(l) \]

\[ E_{\langle \text{interchange} \rangle}(l, r_1, r_2) = \lambda \sigma. \lambda r. \text{let } \{ \{ \ldots \} \} = \sigma(l) \]

Matrix transposition

\[ A = \text{tensor}([N_1, N_2]) \]

\[ B = \text{eq}(A, [i_1, i_2] \rightarrow [i_2, i_1]) \]

\[ E_{\langle \text{build} \rangle}(B)[\sigma] = \{ i, \{ \{ \ldots \} \} \rightarrow \{ \{ \ldots \} \} \} \]

Compositions

Contraction

\[ \mathcal{P}_{stmt}[t_0'] = \text{contract}(t_0, t_1, [r_0, r_1]) \]

Tiling

\[ \text{codegen}([l_1, l_2, l_3, l_4, l_5, l_6]) \]

Experiments

Helmholtz

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

Mttkrp

- 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 -openmp)

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.