Towards Compositional and Generative Tensor Optimizations

Adilla Susungi\textsuperscript{1}, Norman A. Rink\textsuperscript{2}, Jerónimo Castrillón\textsuperscript{2}, Immo Huismann\textsuperscript{3}, Albert Cohen\textsuperscript{4}, Claude Tadonki\textsuperscript{1}, Jörg Stiller\textsuperscript{3} and Jochen Fröhlich\textsuperscript{3}

\textsuperscript{1}MINES ParisTech, PSL Research University
\textsuperscript{2}Chair for Compiler Construction, Technische Universität Dresden
\textsuperscript{3}Chair of Fluid Mechanics, Technische Universität Dresden
\textsuperscript{4}Inria, Ecole normale supérieure

16th International Conference on Generative Programming: Concepts & Experiences (GPCE’17)
Vancouver, Canada, October 24, 2017
Tensor Computations

- Underlying data structure: N-dimensional array

Applications in numerical applications

- Quantum chemistry
- Machine learning
- Big data
- Computational fluid dynamics
Frameworks for Optimizations for Tensor Computations

- Domain-specific expressivity
- Flexible/Adaptive optimization heuristics
- Generic expressivity
- Hidden and/or rigid optimization heuristics
Tensors in Computational Fluid Dynamics

Characteristics

- 3 to 4 dimensions nesting
- Few iterations per dimension (e.g., 13 iterations)
- Tensor contractions, outer products, entrywise multiplications
- Same computation for each element of a mesh

Inverse Helmholtz [7]

\[ t_{ijk} = \sum_{l,m,n} A^T_{kn} \cdot A^T_{jm} \cdot A^T_{il} \cdot u_{lmn} \]

\[ p_{ijk} = D_{ijk} \cdot t_{ijk} \]

\[ v_{ijk} = \sum_{l,m,n} A_{kn} \cdot A_{jm} \cdot A_{il} \cdot p_{lmn} \]
Tensors in Computational Fluid Dynamics

**Characteristics**
- 3 to 4 dimensions nesting
- Few iterations per dimension (e.g., 13 iterations)
- Tensor contractions, outer products, entrywise multiplications
- Same computation for each element of a mesh

**Inverse Helmholtz [7]**

\[
t_{ijk} = \sum_{l,m,n} A_{kn}^T \cdot A_{jm}^T \cdot A_{il}^T \cdot u_{lmn}
\]

\[
p_{ijk} = D_{ijk} \cdot t_{ijk}
\]

\[
v_{ijk} = \sum_{l,m,n} A_{kn} \cdot A_{jm} \cdot A_{il} \cdot p_{lmn}
\]
Implementing CFD Kernels in Existing Frameworks

- Chill • [6]
- Pluto • [5]
- TensorFlow • [3]
- TVM • [2]
- Tensor Contraction Engine • [4]
- Numpy • [1]
- Tensor Algebra Compiler • [8]
Implementing CFD Kernels in Existing Frameworks

We encounter different levels of limitations

- Limited expressivity
- No optimization ability
- Unadapted heuristics
- Unadapted constructs
Our contribution

An intermediate language with building blocks for declaring:

- Tensor computations
- Optimization heuristics

Arrays, tensor operators, iterators and loop transformations as first class citizens.
Our contribution

An intermediate language with building blocks for declaring:

- Tensor computations
- Optimization heuristics

Arrays, tensor operators, iterators and loop transformations as first class citizens.

CFD kernels share common tensor operations with other domains

- We want enough flexibility and genericity (at least for tensor-based applications) to be reused in other domains.
Inverse Helmholtz by Example

\[ t_{ijk} = \sum_{l,m,n} A^T_{kn} \cdot A^T_{jm} \cdot A^T_{il} \cdot u_{lmn} \]

\[ p_{ijk} = D_{ijk} \cdot t_{ijk} \]

\[ v_{ijk} = \sum_{l,m,n} A_{kn} \cdot A_{jm} \cdot A_{il} \cdot p_{lmn} \]

**Step 1: Declaring tensor computations**

\[
\begin{align*}
A &= \text{array}(2, \text{double}, [N, N]) \\
u &= \text{array}(3, \text{double}, [N, N, N]) \\
D &= \text{array}(3, \text{double}, [N, N, N]) \\
At &= \text{vtranspose}(A, 1, 2) \\
tmp1 &= \text{contract}(At, u, [2, 1]) \\
tmp2 &= \text{contract}(At, tmp1, [2, 2]) \\
tmp3 &= \text{contract}(At, tmp2, [2, 3]) \\
tmp4 &= \text{entrywise}(D, tmp3) \\
tmp5 &= \text{contract}(A, tmp4, [2, 1]) \\
tmp6 &= \text{contract}(A, tmp5, [2, 2]) \\
v &= \text{contract}(A, tmp6, [2, 3])
\end{align*}
\]
Inverse Helmholtz by Example

Step 2: Associating iterators to computations

i1 = iterator(0, N, 1)
i2 = iterator(0, N, 1)
# ... other iterator declarations

build(D, [td1, td2, td3])
build(tmp1, [i1, i2, i3, i4])
## Also applies to tmp2, ..., tmp6
build(v, [k12, k22, k32, k42])
Step 3: Applying transformations

interchange(i4, i3)
interchange(i4, i2)
interchange(j2, j1)
interchange(j1, j4)
Inverse Helmholtz by Example

Example of results from different heuristics

- **Variant L1**: Loop interchanges only + parallelization;
- **Variant L2**: Loop interchanges + data transpositions of tensor A + parallelization;
- **Variant L3**: Loop interchanges + data transpositions of tensors tmp1, ..., tmp6 + parallelization.
- **Pluto1**: Loop interchanges + parallelization + vectorization;
- **Pluto2**: Loop interchanges + partial fusions + vectorization;
- **Pluto3**: Loop interchanges + maximum fusions + vectorization;

- **Mesh size**: 750; **data size**: 33.
- **Baseline**: sequential execution.
- **Machine**: 24-core Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz (Haswell)
Conclusion

- Cross-domain building-blocks
  - One intermediate language to rule them all flexibly
- Possibility to assess different variants
  - Through meta-programming or auto-tuning techniques

Ongoing work

- Syntax refinement
- Formal semantics
- Applications to other domains
References I

NumPy, package for scientific computing with Python.

TVM: An End to End IR Stack for Deploying Deep Learning Workloads on Hardware Platforms.

Abadi, M., and et al., A. A.
Tensorflow: Large-scale machine learning on heterogeneous distributed systems.

Synthesis of high-performance parallel programs for a class of ab initio quantum chemistry models.
Proceedings of the IEEE 93, 2 (Feb 2005), 276–292.
References II

Bondhugula, U., Hartono, A., Ramanujam, J., and Sadayappan, P.
A practical automatic polyhedral program optimization system.

Chen, C., Chame, J., and Hall, M.
Chill: A framework for composing high-level loop transformations.

Huismann, I., Stiller, J., and Fröhlich, J.
Factorizing the factorization — a spectral-element solver for elliptic equations with linear operation count.

Kjolstad, F., Kamil, S., Chou, S., Lugato, D., and Amarasinghe, S.
The tensor algebra compiler.
In Proceedings of ACM Program. Lang (October 2017), OOPSLA’ 17.