Towards Compositional and Generative Tensor Optimizations

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

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

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

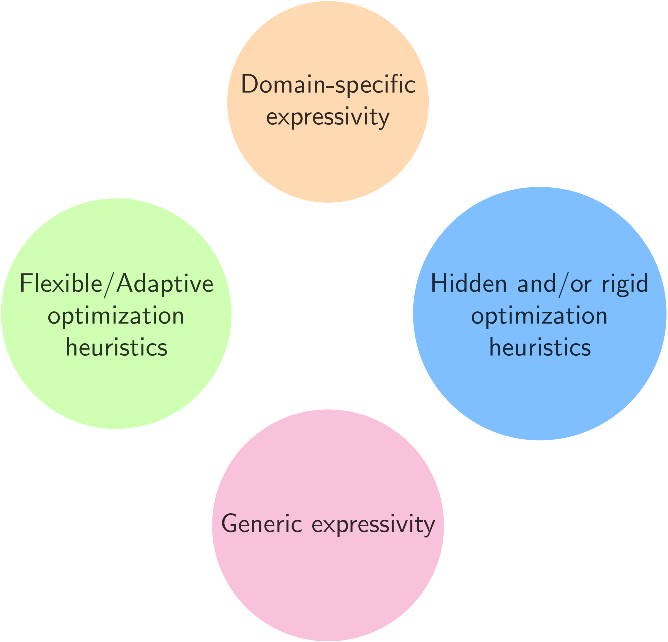
Tensor Computations

- 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



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]

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

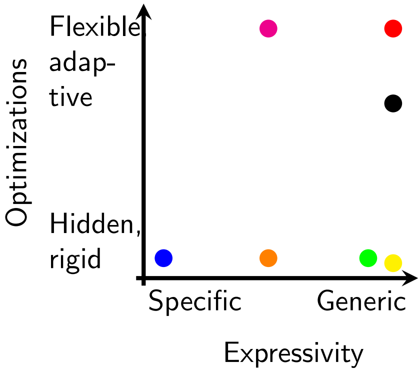
$$p_{ijk} = D_{ijk} \cdot t_{ijk}$$

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

Search space for optimizations may include

- Evaluation order of tensor contractions
- Fusions
- Interchanges
- Transpositions
- Vectorization
- Collapsing
- Unrolling

Implementing CFD Kernels in Existing Frameworks



Chill ● [6]

Pluto ● [5]

TensorFlow ● [3]

TVM ● [2]

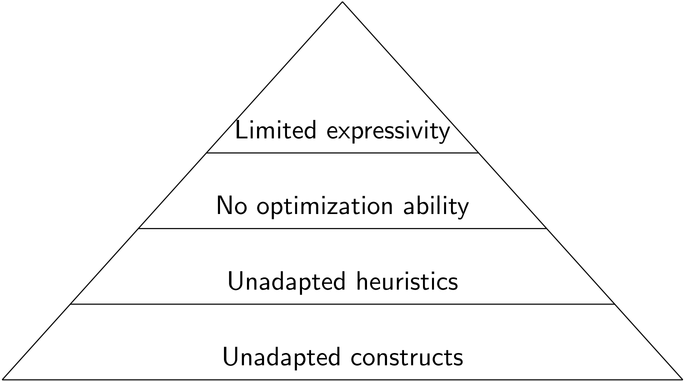
Tensor Contraction Engine ● [4]

Numpy ● [1]

Tensor Algebra Compiler ● [8]

Implementing CFD Kernels in Existing Frameworks

We encounter different levels of limitations

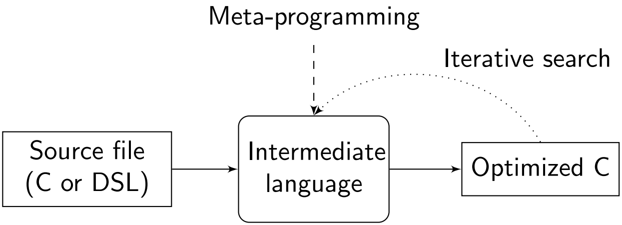


Our contribution

An intermediate language with building blocks for declaring:

- Tensor computations
- Optimization heuristics

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

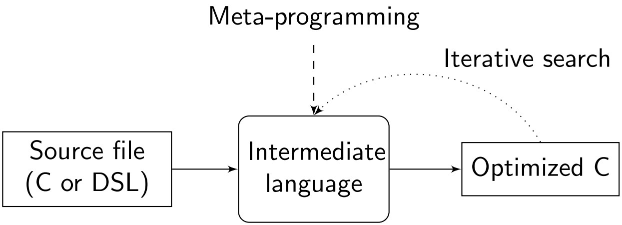


Our contribution

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

```
A = array(2, double, [N, N])
u = array(3, double, [N, N, N])
D = array(3, double, [N, N, N])
At = vtranspose(A, 1, 2)
tmp1 = contract(At, u, [2, 1])
tmp2 = contract(At, tmp1, [2, 2])
tmp3 = contract(At, tmp2, [2, 3])
tmp4 = entrywise(D, tmp3)
tmp5 = contract(A, tmp4, [2, 1])
tmp6 = contract(A, tmp5, [2, 2])
v = contract(A, tmp6, [2, 3])
```

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])
```

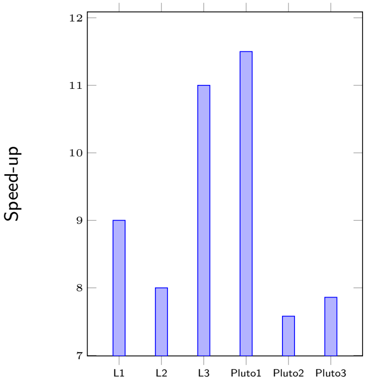
Inverse Helmholtz by Example

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



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

- 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;

Conclusion

- Cross-domain building-blocks

  $\rightarrow$ One intermediate language to rule them all flexibly
- Possibility to assess different variants

  $\rightarrow$ 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.
http://www.numpy.org/, 2017.

TVM: An End to End IR Stack for Deploying Deep Learning Workloads on Hardware Platforms.
https://www.tvmlang.org, 2017.

Abadi, M., and et al., A. A.
Tensorflow: Large-scale machine learning on heterogeneous distributed systems.
http://download.tensorflow.org/paper/whitepaper2015.pdf, 2015.

Baumgartner, G., Auer, A., Bernholdt, D. E., Bibireata, A., Choppella, V., Cociorva, D., Gao, X., Harrison, R. J., Hirata, S., Krishnamoorthy, S., Krishnan, S., chung Lam, C., Lu, Q., Nooijen, M., Pitzer, R. M., Ramanujam, J., Sadayappan, P., and Sibiryakov, A.
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.
In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)* (2008).

Chen, C., Chame, J., and Hall, M.
Chill: A framework for composing high-level loop transformations.
Tech. rep., Technical Report 08-897, University of Southern California, 2008.

Huismann, I., Stiller, J., and Fröhlich, J.
Factorizing the factorization — a spectral-element solver for elliptic equations with linear operation count.
Journal of Computational Physics 346 (2017), 437–448.

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.