A Compiler for Linear Algebra Operations

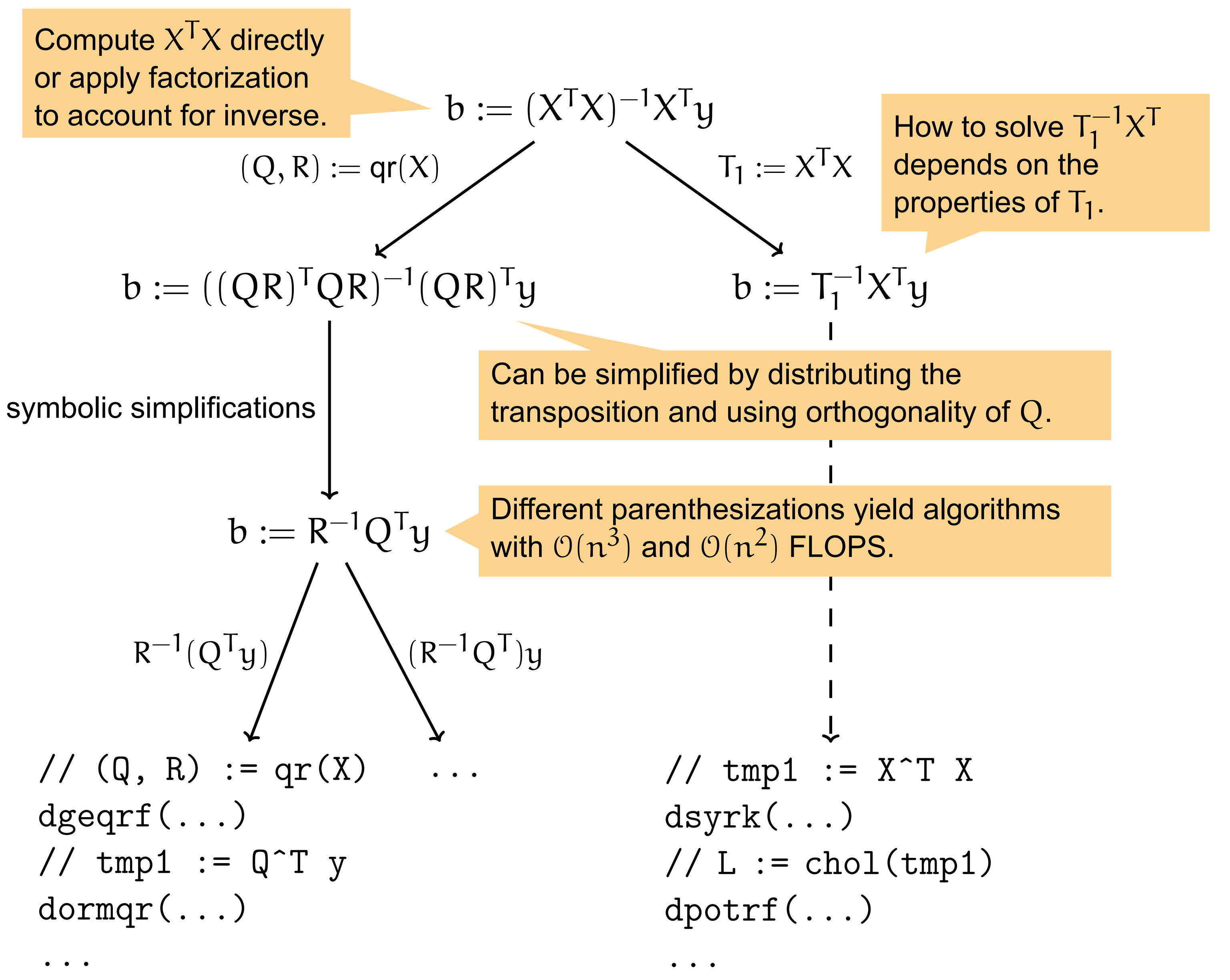
Henrik Barthels

Introduction

- Translating the mathematical description of a linear algebra problem to efficient code is a difficult task. Examples of such problems are:

$$b := (X^TX)^{-1}X^Ty \qquad x_{ij} := A_iBc_j$$
$$x := \left(A^{-T}B^TBA^{-1} + R^T[\Lambda(Rz)]R\right)^{-1} A^{-T}B^TBA^{-1}y$$

- Languages such as Matlab and Julia are easy to use, but performance is usually suboptimal.
- We are developing a compiler that offers the ease-of-use, and thus, productivity of Matlab paired with performance that comes close to what a human expert achieves.



Input Grammar

Unlike Matlab, the input language allows to annotate operands with properties.

definitions → definition$^+$
definition → type *name* size ⟨property*⟩
type → **Matrix** | **Vector** | **Scalar**
size → (*rows*, *columns*)
property → **LowerTriangular** | **Orthogonal** | ...

assignments → assignment$^+$
assignment → operand := expr
expr → operand | expr + expr | expr · expr | expr^{-1} | expr^T | expr^{-T}
operand → *symbol* | *symbol*$_{\text{indices}}$
indices → *index*$^+$

Instruction Set

As the instruction set, we use optimized kernels as offered by linear algebra libraries.

BLAS [3]

- $y \leftarrow \alpha Ax + \beta y$
- $C \leftarrow \alpha AB + \beta C$
- $B \leftarrow \alpha A^{-1}B$
- ...

LAPACK [1]

- Matrix factorizations.
- Eigensolvers.
- Solvers for linear systems with specific properties.

Optimizations

Common Subexpression Elimination

- FLOPS can be saved by reusing subexpressions that appear more than once, for example AB in $AB + ABC$.
- Operators such as transposition and inversion make the detection of common subexpression more complicated:

$$\underbrace{AB^{-T}}_{C_1}\underbrace{B^{-1}A^T}_{C_2} \to C_1C_1^T \qquad C_1 = AB^{-T} = \left(B^{-1}A^T\right)^T = C_2^T$$

- We developed algorithms to detect such common subexpressions.

Generalized Matrix Chain Problem

- The cost of a product $M_0 \cdots M_n$ highly depends on the parenthesization.
- We developed a generalized version of the $\mathcal{O}(n^3)$ matrix chain algorithm [2].
- It finds the optimal parenthesization for matrix chains containing transposition and inversion, for example $X := AB^{-T}C^{-1}D$.
- This algorithm also takes properties into account.

Encoded Linear Algebra Knowledge

Knowledge about linear algebra is used to simplify and rewrite expressions, as well as to infer properties. Matrix properties are crucial to select the most suitable kernels, as well as for simplifications.

Inference of Properties

$A \to A^T$
$A \;\; B \to AB$
$A \;\; B \to AB$
$\text{SPD}(S) \to \text{SPD}(S^{-1})$
$\text{expr} = \text{expr}^T \to \text{Symmetric}(\text{expr})$

Simplifications

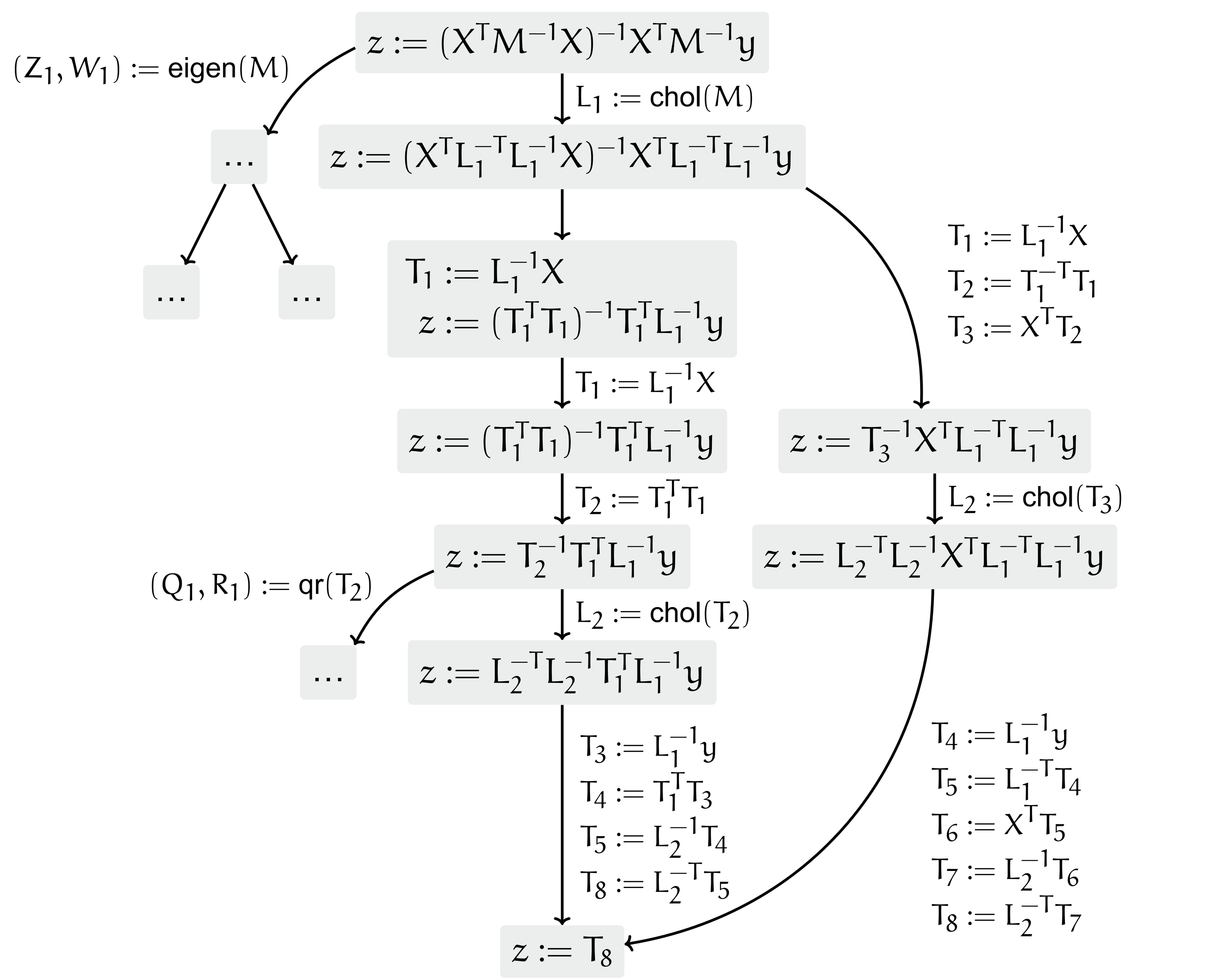
$(AB)^T \to B^TA^T$
$A^T \to A$ if Symmetric(A)
$Q^TQ \to I$ if Orthogonal(Q)
$Q^{-1} \to Q^T$ if Orthogonal(Q)
$\alpha A + \beta A \to (\alpha + \beta)A$

Derivation Graph

Algorithms are derived by repeatedly applying different derivation steps.

Input → Simplifications → Common Subexpr. - - → Compute → Algorithms

Each of these steps yields one or more results. We use a graph to represent all algorithms. The root of the graph is the input expression, and each path in the graph corresponds to one algorithm. Below, a part of such a graph is shown. The full graph has 83 nodes.



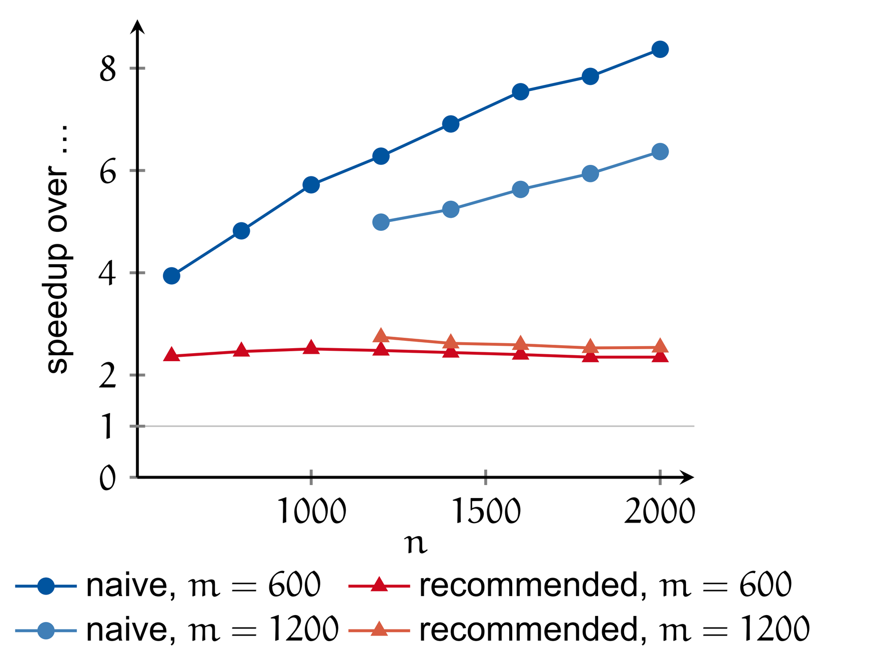
Results

- Example: $z := (X^TM^{-1}X)^{-1}X^TM^{-1}y$, $M \in \mathbb{R}^{n\times n}$, $X \in \mathbb{R}^{n\times m}$, $y \in \mathbb{R}^n$, $n \geqslant m$. M is symmetric positive definite.
- The compiler (written in Python) finds 23 algorithms in less than a second.
- The algorithm shown below is identified as the cheapest one in terms of FLOPS.
- It is compared to two other implementations that rely on Matlab's strategy to evaluate expressions.

Naive implementation
```
z = inv(X'*inv(M)*X)*X'*inv(M)*y;
```

Recommended implementation
```
z = (X'*(M\X))\X'*(M\y);
```

Compiler implementation
```
L1 = chol(M, 'lower');
t1 = linsolve(L1, X, opts1);
t2 = t1'*t1;
L2 = chol(t2, 'lower');
t3 = linsolve(L1, y, opts1);
t4 = t1'*t3;
t5 = linsolve(L2, t4, opts1);
z = linsolve(L2, t5, opts2);
```



References

[1] E. Anderson, Z. Bai, et al. *LAPACK Users' guide*, volume 9. SIAM, 1999.
[2] T. H. Cormen, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill, Inc., 1990.
[3] J. J. Dongarra, J. Du Croz, et al. A set of Level 3 Basic Linear Algebra Subprograms. *ACM TOMS*, 16(1):1–17, 1990.

Henrik Barthels
Aachen Institute for Advanced Study in Computational Engineering Science (AICES)
RWTH Aachen, Germany
http://hpac.rwth-aachen.de
barthels@aices.rwth-aachen.de

HPAC High Performance and Automatic Computing

RWTH AACHEN UNIVERSITY