The Linear Algebra Mapping Problem
and how programming languages solve it

Paolo Bientinesi
pauldj@cs.umu.se

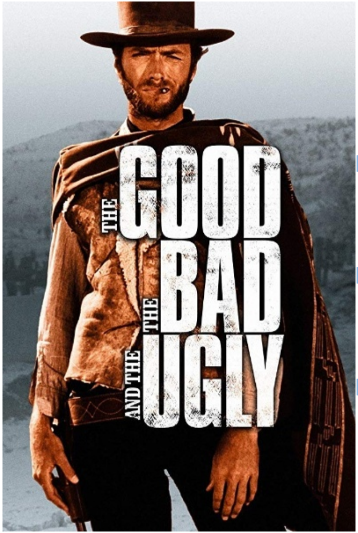
September 12, 2023
NHR PerfLab Seminar

HPAC High Performance and Automatic Computing

UMEÅ UNIVERSITY

HPC2N

Sneak preview



- GOOD: Plenty of excellent libraries for matrix computations
- BAD: Are they used (properly)? Not so much
- UGLY: Current state of high-level programming languages

Applications

Signal Processing	$x := \left(A^{-T}B^TBA^{-1} + R^TLR\right)^{-1}A^{-T}B^TBA^{-1}y$	$R \in \mathbb{R}^{n-1\times n}$, UT; $L \in \mathbb{R}^{n-1\times n-1}$, DI
Kalman Filter	$K_k := P_k^bH^T(HP_k^bH^T + R)^{-1};\ x_k^a := x_k^b + K_k(z_k - Hx_k^b);\ P_k^a := (I - K_KH)\,P_k^b$	
Ensemble Kalman Filter	$X^a := X^b + \left(B^{-1} + H^TR^{-1}H\right)^{-1}\left(Y - HX^b\right)$	$B \in \mathbb{R}^{N\times N}$ SSPD; $R \in \mathbb{R}^{m\times m}$, SSPD
Image Restoration	$x_k := (H^TH + \lambda\sigma^2I_n)^{-1}(H^Ty + \lambda\sigma^2(v_{k-1} - u_{k-1}))$	
Rand. Matrix Inversion	$X_{k+1} := S(S^TAS)^{-1}S^T + (I_n - S(S^TAS)^{-1}S^TA)X_k(I_n - AS(S^TAS)^{-1}S^T)$	
Generalized Least Squares	$b := (X^TM^{-1}X)^{-1}X^TM^{-1}y$	$n > m$; $M \in \mathbb{R}^{n\times n}$, SPD; $X \in \mathbb{R}^{n\times m}$; $y \in \mathbb{R}^{n\times 1}$
Stochastic Newton	$B_k := \frac{k}{k-1}B_{k-1}(I_n - A^TW_k((k-1)I_l + W_k^TAB_{k-1}A^TW_k)^{-1}W_k^TAB_{k-1})$	
Optimization	$x_f := WA^T(AWA^T)^{-1}(b - Ax);\quad x_o := W(A^T(AWA^T)^{-1}Ax - c)$	
Triangular Matrix Inv.	$X_{10} := L_{10}L_{00}^{-1};\quad X_{20} := L_{20} + L_{22}^{-1}L_{21}L_{11}^{-1}L_{10};\quad X_{11} := L_{11}^{-1};\quad X_{21} := -L_{22}^{-1}L_{21}$	
Tikhonov Regularization	$x := (A^TA + \Gamma^T\Gamma)^{-1}A^Tb$	$A \in \mathbb{R}^{n\times m}$; $\Gamma \in \mathbb{R}^{m\times m}$; $b \in \mathbb{R}^{n\times 1}$
Gen. Tikhonov reg.	$x := x_0 + (A^TPA + Q)^{-1}(A^TP(b - Ax_0))$	
LMMSE estimator	$K_{t+1} := C_tA^T(AC_tA^T + C_z)^{-1};\ x_{t+1} := x_t + K_{t+1}(y - Ax_t);\ C_{t+1} := (I - K_{t+1}A)C_t$	

Wealth of excellent LA libraries

PETSc, Trilinos, ...

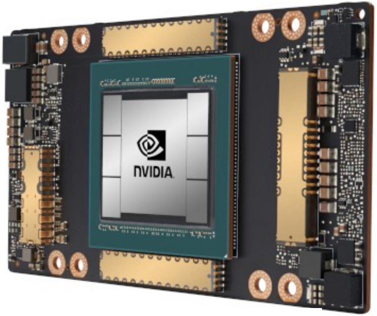
ScaLAPACK, PLAPACK, Elemental, ...

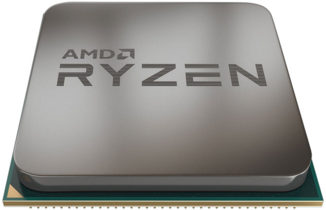
LAPACK, Plasma, SuperMatrix, Magma, ...

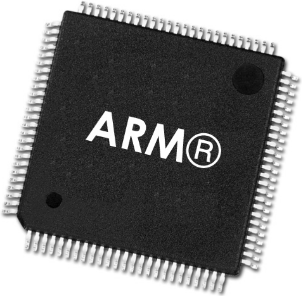
BLAS-1, BLAS-2, BLAS-3, ATLAS, BTO-BLAS, BLIS, ...

... and many more sparse and iterative ones


INTEL® CORE™ i9
i9-10900K
SRH91 3.70GHZ
X016D812 e4


NVIDIA


AMD
RYZEN


ARM®

But . . . discrepancy between building blocks and applications

- BLAS: **Basic** Linear Algebra Subprograms

  $\boldsymbol{w} := \boldsymbol{x}^T \boldsymbol{y}$ (DOT), $\boldsymbol{y} := \alpha \boldsymbol{x} + \boldsymbol{y}$ (AXPY), $\eta := (\boldsymbol{x}^T \boldsymbol{x})^{1/2}$ (NORM)

  $\boldsymbol{y} := \boldsymbol{A}\boldsymbol{x}$ (GEMV), $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ (TRSV)

  $\boldsymbol{C} := \boldsymbol{A}\boldsymbol{B}$ (GEMM), $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B}$ (TRSM)

- LAPACK: Linear Algebra Package

$\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}$	eigenproblems
$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$	linear systems, least squares problems
$\boldsymbol{Q}\boldsymbol{R} = \boldsymbol{A}$	matrix factorizations, . . .

and . . . discrepancy between notation and code

- Mathematical expression: $\boldsymbol{C} := \boldsymbol{C} + \boldsymbol{A} * \boldsymbol{B}^T + \boldsymbol{B} * \boldsymbol{A}^T$

- Corresponding “naive” code:

```
CALL DGEMM( 'N', 'Y', N-K-KB+1, N-K-KB+1, KB, ONE, A( K+KB, K+KB ), LDA,
      $     B( K, K+KB ), LDB, ONE, C( K+KB, K+KB ), LDC )
CALL DGEMM( 'N', 'Y', N-K-KB+1, N-K-KB-1, KB, ONE, B( K+KB, K+KB ), LDA,
      $     A( K, K+KB ), LDB, ONE, C( K+KB, K+KB ), LDC )
```

- The “right” call:

```
CALL DSYR2K( UPLO, 'Transpose', N-K-KB+1, KB, -ONE, A( K, K+KB ), LDA,
      $      B( K, K+KB ), LDB, ONE, A( K+KB, K+KB ), LDA )
```

- 12 args, explicit indexing, not user friendly. Certainly not users' preference anymore.

Users' preference?   High-level languages

$$\boldsymbol{C} := \boldsymbol{C} + \boldsymbol{A} * \boldsymbol{B}^T + \boldsymbol{B} * \boldsymbol{A}^T$$

```
C = A * B'+ B * A' + C;                        // Matlab, Octave

C = A * transpose(B) + B * transpose(A) + C    // Julia

C = A * trans(B) + B * trans(A) + C;           // Armadillo

C = A * B.transpose() + B * A.transpose() + C; // Eigen

ct = at @ bt.T + bt @ at.T + ct                // NumPy

ct <- at %*% t(bt) + bt %*% t(at) + ct         // R

C = A@tf.transpose(B) + B@tf.transpose(A) + C  // TensorFlow

C = A@torch.t(B) + B@torch.t(A) + C            // PyTorch
```

$$K_k := P_k^b H^T (H P_k^b H^T + R)^{-1}; \quad x_k^a := x_k^b + K_k (z_k - H x_k^b); \quad P_k^a := (I - K_K H)\, P_k^b$$

$$\begin{cases} C_\dagger := PCP^T + Q \\ K := C_\dagger H^T (H C_\dagger H^T)^{-1} \end{cases}$$

$$\Lambda := S(S^T AWAS)^{-1} S^T; \; \Theta := \Lambda A W; \; M_k := X_k A - I$$
$$X_{k+1} := X_k - M_k \Theta - (M_k \Theta)^T + \Theta^T (A X_k A - A) \Theta$$

$$x := A(B^T B + A^T R^T \Lambda R A)^{-1} B^T B A^{-1} y \qquad \ldots \qquad E := Q^{-1} U (I + U^T Q^{-1} U)^{-1} U^T$$

?

$y := \alpha x + y$ | $LU = A$ | $\cdots$ | $C := \alpha AB + \beta C$

$X := A^{-1} B$ | $C := AB^T + BA^T + C$ | $X := L^{-1} M L^{-T}$ | $QR = A$

$$K_k := P_k^b H^T (H P_k^b H^T + R)^{-1}; \quad x_k^a := x_k^b + K_k(z_k - H x_k^b); \quad P_k^a := (I - K_K H)\, P_k^b$$

$$\begin{cases} C_\dagger := PCP^T + Q \\ K := C_\dagger H^T (H C_\dagger H^T)^{-1} \end{cases}$$

$$\Lambda := S(S^T AWAS)^{-1} S^T; \; \Theta := \Lambda AW; \; M_k := X_k A - I$$
$$X_{k+1} := X_k - M_k \Theta - (M_k \Theta)^T + \Theta^T (A X_k A - A)\Theta$$

$$x := A(B^T B + A^T R^T \Lambda R A)^{-1} B^T B A^{-1} y \quad \ldots \quad E := Q^{-1} U (I + U^T Q^{-1} U)^{-1} U^T$$

LINEAR ALGEBRA
MAPPING PROBLEM
"LAMP"

$y := \alpha x + y$ | $LU = A$ | $\cdots$ | $C := \alpha AB + \beta C$

$X := A^{-1}B$ | $C := AB^T + BA^T + C$ | $X := L^{-1} M L^{-T}$ | $QR = A$

All the aforementioned high-level languages solve LAMPs

Linear Algebra Mapping Problem (LAMP)

Relevant to both dense and sparse computations

- $\mathcal{E}$: a sequence of explicit assignments $\boldsymbol{var_i := EXP_i}$
- $\mathcal{K}$: a set of available computational building blocks e.g., BLAS, LAPACK, ...
- $\mathcal{M}$: a cost function defined over $\mathcal{K}^+$ #FLOPs, exec. time, #mem.ops, stability

LAMP

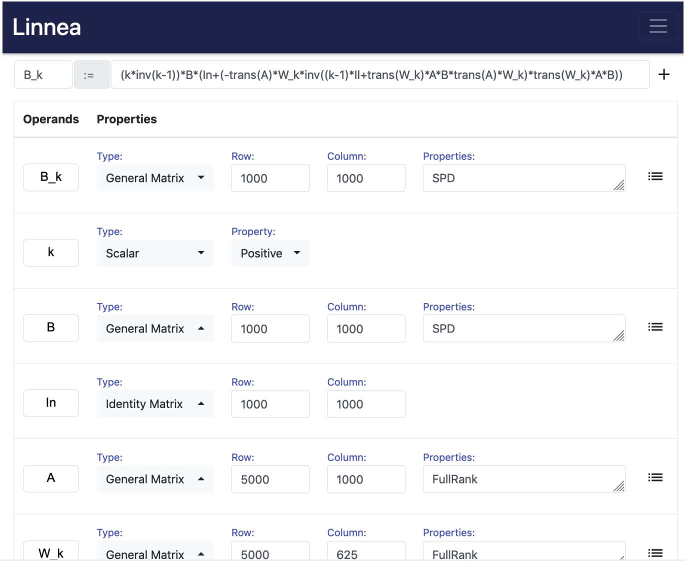
Find a sequence of calls to building blocks in $\mathcal{K}$, optimal according to $\mathcal{M}$, that computes all the assignments in $\mathcal{E}$.

- Suboptimal solution $\rightarrow$ easy
- Optimality $\rightarrow$ NP complete $\leftarrow$ reduction from Ensemble Computation

C. Psarras, H. Barthels, P. Bientinesi, [arXiv:1911.09421]
"The Linear Algebra Mapping Problem. Current state of linear algebra languages and libraries", ACM TOMS, 2022

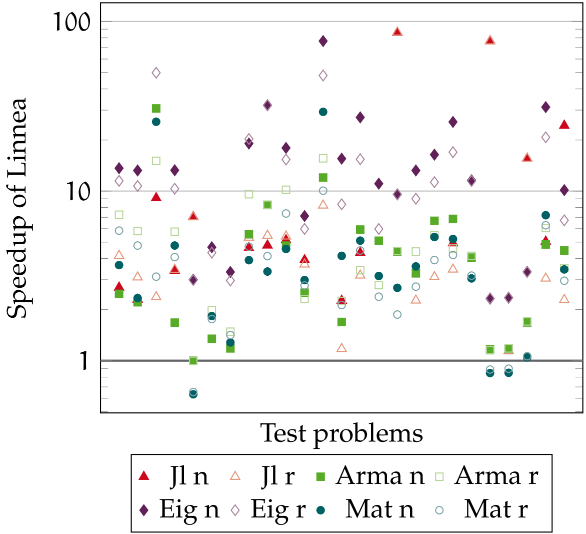
Linnea — https://linnea.cs.umu.se/

Henrik Barthels

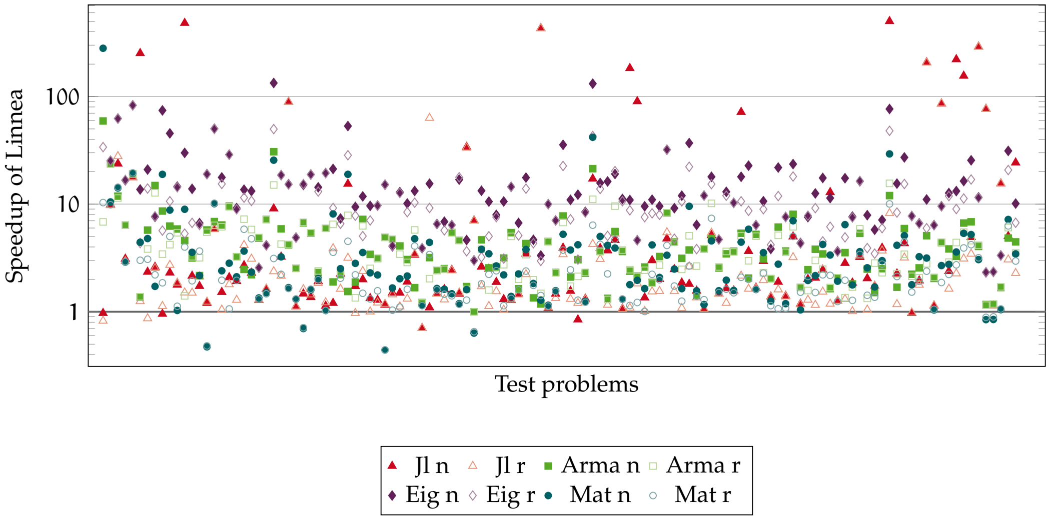


H. Barthels, C. Psarras, P. Bientinesi, [arXiv:1912.12924]
"Linnea: Automatic Generation of Efficient Linear Algebra Programs", ACM TOMS, 2021

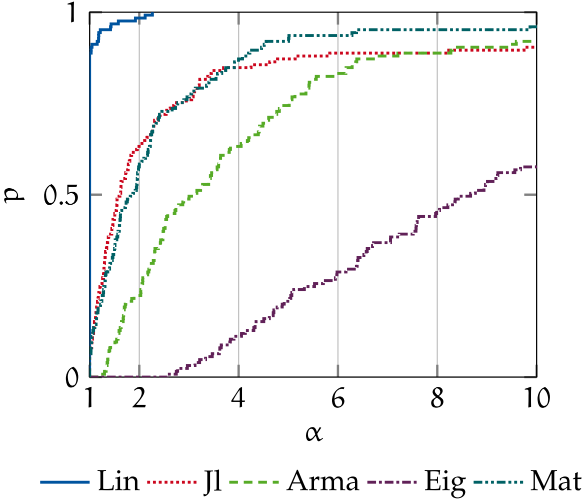
Linnea vs. languages — Application problems



Linnea vs. languages — Random problems

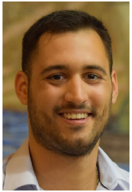


Linnea vs. languages — Performance profiles



Investigation: How well do high-level languages solve LAMPs?

1. Matlab
2. Octave
3. Julia
4. C++ with Armadillo
5. C++ with Eigen
6. NumPy
7. R
8. (TensorFlow)
9. (PyTorch)



Christos Psarras

Aravind Sankaran

- 12 experiments, each exposing one specific optimization
- Not a ranking of languages!

- "**The Linear Algebra Mapping Problem. Current state of linear algebra languages and libraries**", ACM Transactions on Mathematical Software, Vol. 48(3), pp.1–30, September 2022. [arXiv:1911.09421]
- *"Benchmarking the Linear Algebra Awareness of TensorFlow and PyTorch"*, Proceedings of iWAPT-22. [arXiv:2202.09888]

Q1: Do they map? matrix products

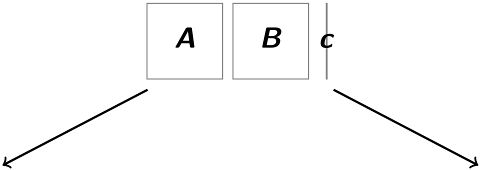
input	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
C = A*B	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27
GEMM	✓	✓	✓	✓	✓	✓	✓	
C = C+A*A'	0.18	0.17	0.21	0.32	0.29	0.17	0.18	0.14
SYRK	✓	✓	✓	×	×	✓	✓	
C = C+AB'+BA'	0.57	0.59	0.69	0.59	0.58	0.57	0.58	0.28
SYR2K	×	×	×	×	×	×	×	

Do they map? $\boldsymbol{Ax = b} \quad \equiv \quad \boldsymbol{x := A \backslash b} \quad \not\equiv \quad \mathrm{inv}(\boldsymbol{A}) * \boldsymbol{b}$

`input`	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
$\boldsymbol{x := A \backslash b}$	0.71	0.72	0.63	0.68	0.64	0.63	0.72	0.61
$\mathrm{inv}(\boldsymbol{A}) * \boldsymbol{b}$	1.76	1.82	1.69	2.20	2.21	**0.63**	2.49	1.71
LinSolve	-	-	-	-	-	✓	-	-

but . . . should they map?

Optimal parenthesisation



$(\boldsymbol{AB})\boldsymbol{c} \quad \mathcal{O}(n^3)$ $\qquad$ $\boldsymbol{A}(\boldsymbol{Bc}) \quad \mathcal{O}(n^2)$

Matrix product is associative, but its cost is not

$$\boldsymbol{ABCD} = (\boldsymbol{A}(\boldsymbol{B}(\boldsymbol{CD}))) = (\boldsymbol{A}((\boldsymbol{BC})\boldsymbol{D})) = (\boldsymbol{AB})(\boldsymbol{CD}) = (((\boldsymbol{AB})\boldsymbol{C})\boldsymbol{D}) = ((\boldsymbol{A}(\boldsymbol{BC}))\boldsymbol{D})$$

BUT the best parenthesisation depends on the sizes of the matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$

Q2: Matrix Chain?

Chain	**Optimal Evaluation**
1) "left-to-right"	((A B) C)
2) "right-to-left"	(A (B C))
3) "mixed"	((A B) (C D))

		Matlab	Octave	Julia	R	Eigen	Armad.	NumPy
1)	(A*B)*C	0.056	0.056	0.055	0.061	0.058	0.056	0.055
	A*B*C	0.056	0.056	0.055	0.061	0.058	0.056	0.055
2)	A*(B*C)	0.055	0.056	0.054	0.059	0.056	0.055	0.056
	A*B*C	0.42	0.43	0.42	0.44	0.42	**0.055**	0.42
3)	(A*B)*(C*D)	0.21	0.22	0.22	0.22	0.23	0.20	0.22
	A*B*C*D	0.32	0.33	0.33	0.33	0.35	0.31	0.33
	Matrix chains	×	×	×	×	×	≈	×

Wait! New release!

Chain	**Optimal Evaluation**
1) “left-to-right”	((A B) C)
2) “right-to-left”	(A (B C))
3) “mixed”	((A B) (C D))

		Matlab	Octave	**Julia**	R	Eigen	Armad.	NumPy
1)	(A*B)*C	0.056	0.056	0.055	0.061	0.058	0.056	0.055
	A*B*C	0.056	0.056	0.055	0.061	0.058	0.056	0.055
2)	A*(B*C)	0.055	0.056	0.054	0.059	0.056	0.055	0.056
	A*B*C	0.42	0.43	**0.054**	0.44	0.42	**0.055**	0.42
3)	(A*B)*(C*D)	0.21	0.22	0.22	0.22	0.23	0.20	0.22
	A*B*C*D	0.32	0.33	**0.22**	0.33	0.35	0.31	0.33
	Matrix chains	×	×	✓	×	×	≈	×

Challenge: Generalized Matrix Chain

[arXiv:1804.04021]

In practice

- Unary operators: transposition, inversion $(\boldsymbol{X} := \boldsymbol{A}\boldsymbol{B}^T\boldsymbol{C}^{-T}\boldsymbol{D} + \dots)$
- Overlapping kernels (e.g., $\boldsymbol{L} \leftarrow \boldsymbol{L}^{-1}$, $\boldsymbol{X} = \boldsymbol{A}^{-1}\boldsymbol{B}$)
- Decompositions (e.g., $\boldsymbol{A} \to \boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{Q}$, $\boldsymbol{A} \to \boldsymbol{L}\boldsymbol{U}$)
- Properties & specialized kernels (`GEMM`, `TRMM`, `SYMM`, ...)

Q3–4: Properties?

Operation	Property	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
A*X = B	-	0.71	0.74	0.62	0.67	0.63	0.62	0.65	0.61
	Symmetric	0.71	0.73	0.62	0.69	N/A	0.62	0.65	0.46
	SPD	**0.41**	**0.40**	0.60	0.63	N/A	**0.34**	0.62	0.31
	Triangular	**0.03**	**0.04**	**0.03**	0.63	N/A	0.62	0.65	0.03
	Diagonal	0.03	0.05	**0.01**	0.63	N/A	**0.03**	0.62	0.001
		$\approx$	$\approx$	$\approx$	$\times$	$\times$	$\approx$	$\times$	
C = A*B	-	1.44	1.48	1.47	1.47	1.45	1.44	1.44	1.46
	Triangular	1.44	1.48	**0.75**	1.47	1.45	1.44	1.44	0.74
	Diagonal	1.44	1.48	**0.03**	1.47	1.45	1.42	1.44	0.06
		$\times$	$\times$	✓	$\times$	$\times$	$\times$	$\times$	

Q5: Common Subexpressions?

$$\begin{cases} \boldsymbol{X} := \boldsymbol{AB} \\ \boldsymbol{Y} := \boldsymbol{AB} \end{cases} \rightarrow \begin{cases} \boldsymbol{X} := \boldsymbol{AB} \\ \boldsymbol{Y} := \boldsymbol{X} \end{cases}$$

	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy
copy	0.27	0.31	0.36	0.30	0.30	0.26	0.30
direct	0.54	0.6	0.61	0.56	0.58	0.52	0.55
	×	×	×	×	×	×	×

Challenge: Common Subexpressions

[arXiv:1710.06915]

$$\begin{cases} X := AB^{-T}C \\ Y := B^{-1}A^{T}D \end{cases} \rightarrow \begin{cases} Z := AB^{-T} \\ X := ZC \\ Y := Z^{T}D \end{cases}$$

BUT

$$X := ABABv \quad \not\rightarrow \quad \begin{cases} Z := AB \\ X := ZZv \end{cases}$$

Q6–8: Other features?

- Code motion

```
for i = 1:n,
    C = A*B;
    d[i] = C[i,i];
end
```

$\rightarrow$?

```
C = A*B;
for i = 1:n,
    d[i] = C[i,i];
end
```

$\times$

- Blocked operands

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M_1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M_2} \end{bmatrix} \rightarrow ? \quad \begin{cases} \boldsymbol{M_1 x_T = y_T} \\ \boldsymbol{M_2 x_B = y_B} \end{cases}, \quad \begin{cases} \boldsymbol{y_T := M_1 x_T} \\ \boldsymbol{y_B := M_2 x_B} \end{cases} \quad \times, \times$$

- $\text{diag}(\boldsymbol{A} + \boldsymbol{B})$ vs. $\text{diag}(\boldsymbol{A}) + \text{diag}(\boldsymbol{B})$ $\rightarrow$ Armadillo ✓

  $\text{diag}(\boldsymbol{AB})$ vs. . . . $\rightarrow$ $\times$

Summary

- LAMPs are challenging: (quasi-)optimal solutions require expertise in LA and HPC
- To users:
  Beware! Compilers & languages are great with scalars, not so much with matrices
- To language developers: Please pay attention to the optimizations we exposed. They arise frequently in the solution of LAMPs.
- Natural –but challenging– extensions to this study:
  - Other optimizations
  - Parallelism
  - Sparse operands

Thank you for your attention!