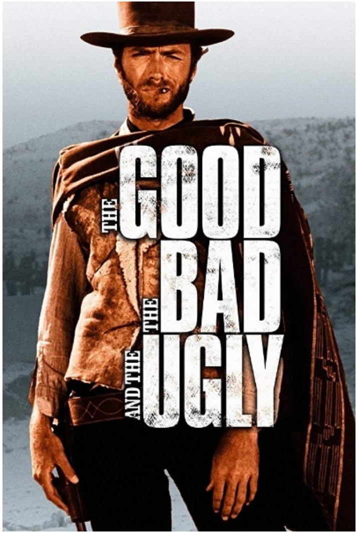
Current state of programming languages for linear algebra computations

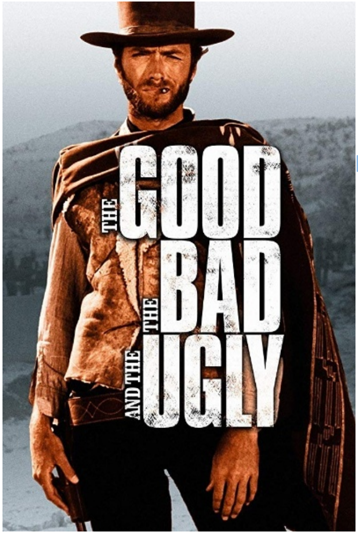
Paolo Bientinesi
pauldj@cs.umu.se

DCSE Summerschool
Numerical Linear Algebra on High Performance Computers
June 07, 2023
TU Delft

Preview

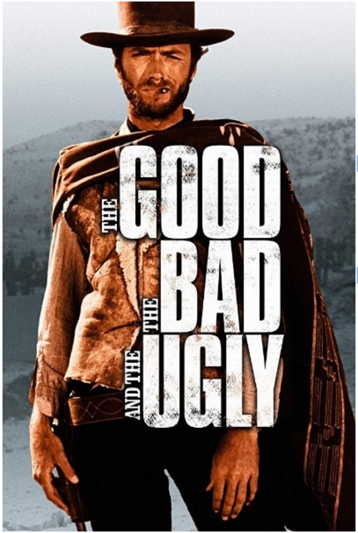


Preview



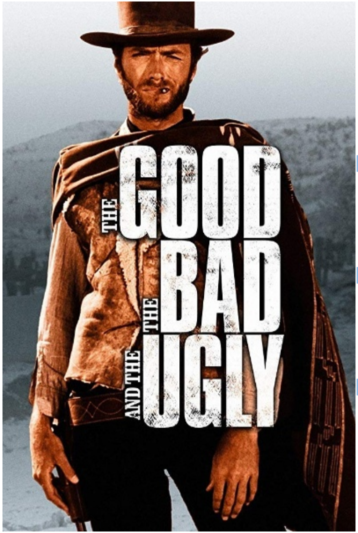
- GOOD: Plenty of excellent libraries for matrix computations

Preview



- GOOD: Plenty of excellent libraries for matrix computations
- BAD: Are they used (properly)? Not so much

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

Hierarchy of Libraries: 1979–1990

“Basic Linear Algebra Subprograms”

BLAS-1, BLAS-2, BLAS-3

Hierarchy of Libraries: 1990s

Solvers & eigensolvers, 1992

LAPACK

BLAS-1, BLAS-2, BLAS-3

Hierarchy of Libraries: 1990s

Distributed memory, 1995–

ScaLAPACK, PLAPACK, . . .

LAPACK

BLAS-1, BLAS-2, BLAS-3

Hierarchy of Libraries: 1990s

Dense & sparse, 1997

PETSc, ...

ScaLAPACK, PLAPACK, ...

LAPACK

BLAS-1, BLAS-2, BLAS-3

Many libraries! . . . and many more sparse and iterative ones

PETSc, Trilinos, . . .

ScaLAPACK, PLAPACK, Elemental, . . .

LAPACK, Plasma, SuperMatrix, Magma, . . .

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

2021 ACM Turing Award

JACK DONGARRA

United States – 2021

CITATION

For his pioneering contributions to numerical algorithms and libraries that enabled high performance computational software to keep pace with exponential hardware improvements for over four decades

RESEARCH SUBJECTS

But . . . issue #1: expression vs. code

But . . . issue #1: expression vs. code

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

But . . . issue #1: expression vs. code

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

- 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 )
```

But . . . issue #1: expression vs. code

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

- 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+1 ), LDB, ONE, A( K+KB, K+KB ), LDA )
```

But . . . issue #1: expression vs. code

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

- 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+1 ), LDB, ONE, A( K+KB, K+KB ), LDA )
```

- 12 args, explicit indexing, not user friendly. Certainly not users’ preference (anymore).

But . . .   issue #1: expression vs. code

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

- 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+1 ), LDB, ONE, A( K+KB, K+KB ), LDA )
```

- 12 args, explicit indexing, not user friendly. Certainly not users' preference (anymore). Did you catch the indexing mistake?

Users' preference(s)?

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

```
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
```

Issue #2: building blocks vs. 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, ...

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	$x_k := (H^TH + \lambda\sigma^2 I_n)^{-1}(H^Ty + \lambda\sigma^2(v_{k-1} - u_{k-1}))$	
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	$K_{t+1} := C_tA^T(AC_tA^T + C_z)^{-1};\ x_{t+1} := x_t + K_{t+1}(y - Ax_t);\ C_{t+1} := (I - K_{t+1}A)C_t$	

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

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

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

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

$$\Downarrow \; ?$$

$$\boxed{y := \alpha x + y} \; \boxed{LU = A} \; \cdots \; \boxed{C := \alpha AB + \beta C}$$

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

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

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

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

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

**LINEAR ALGEBRA
MAPPING PROBLEM
"LAMP"**

$\boxed{y := \alpha x + y}$ $\boxed{LU = A}$ $\cdots$ $\boxed{C := \alpha AB + \beta C}$

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

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

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

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

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

LINEAR ALGEBRA MAPPING PROBLEM “LAMP”

$\boxed{y := \alpha x + y}$ $\boxed{LU = A}$ $\cdots$ $\boxed{C := \alpha AB + \beta C}$

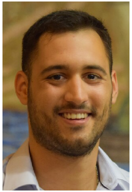
$\boxed{X := A^{-1} B}$ $\boxed{C := AB^T + BA^T + C}$ $\boxed{X := L^{-1} M L^{-T}}$ $\boxed{QR = A}$

All the aforementioned high-level languages solve LAMPs

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

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								

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

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`	✓	✓	✓	✓	✓	✓	✓	

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'								

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

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`	✓	✓	✓	×	×	✓	✓	

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'								

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

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
$\boldsymbol{x := A \backslash b}$								

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

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

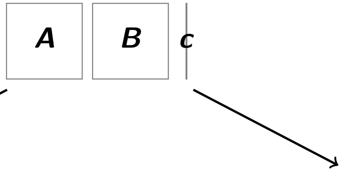
`input`	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
$\boldsymbol{x := A \backslash b}$	0.71	0.72	0.63	0.68	0.64	0.63	0.72	0.61
$\mathrm{inv}(\boldsymbol{A}) * \boldsymbol{b}$	1.76	1.82	1.69	2.20	2.21	**0.63**	2.49	1.71

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

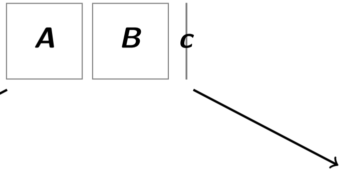


$(\boldsymbol{AB})\boldsymbol{c} \quad \mathcal{O}(n^3)$

$\boldsymbol{A}(\boldsymbol{Bc}) \quad \mathcal{O}(n^2)$

Matrix product is associative, but its cost is not

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))

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

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	0.42	0.43	0.42	0.44	0.42	**0.055**	0.42
	A*(B*C)	0.055	0.056	0.054	0.059	0.056	0.055	0.056

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	0.42	0.43	0.42	0.44	0.42	**0.055**	0.42
	A*(B*C)	0.055	0.056	0.054	0.059	0.056	0.055	0.056
3)	A*B*C*D	0.32	0.33	0.33	0.33	0.35	0.31	0.33
	(A*B)*(C*D)	0.21	0.22	0.22	0.22	0.23	0.20	0.22

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	0.42	0.43	0.42	0.44	0.42	**0.055**	0.42
	A*(B*C)	0.055	0.056	0.054	0.059	0.056	0.055	0.056
3)	A*B*C*D	0.32	0.33	0.33	0.33	0.35	0.31	0.33
	(A*B)*(C*D)	0.21	0.22	0.22	0.22	0.23	0.20	0.22
	Matrix chains	$\times$	$\times$	$\times$	$\times$	$\times$	$\approx$	$\times$

Wait!

Wait! They read our paper!

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	0.42	0.43	**0.054**	0.44	0.42	**0.055**	0.42
	A*(B*C)	0.055	0.056	0.054	0.059	0.056	0.055	0.056
3)	A*B*C*D	0.32	0.33	**0.22**	0.33	0.35	0.31	0.33
	(A*B)*(C*D)	0.21	0.22	0.22	0.22	0.23	0.20	0.22
	Matrix chains	×	×	✓	×	×	≈	×

Challenge: Generalized Matrix Chain

[arXiv:1804.04021]

In practice

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

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

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

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

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
		$\approx$	$\approx$	$\approx$	$\times$	$\times$	$\approx$	$\times$	

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
		≈	≈	≈	×	×	≈	×	
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
		×	×	✓	×	×	×	×	

Q5: Common Subexpressions?

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

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
	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$

Challenge: Common Subexpressions

[arXiv:1710.06915]

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

Challenge: Common Subexpressions [arXiv:1710.06915]

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

BUT

$$\boldsymbol{X} := \boldsymbol{A}\boldsymbol{B}\boldsymbol{A}\boldsymbol{B}\boldsymbol{v} \not\rightarrow \begin{cases} \boldsymbol{Z} := \boldsymbol{A}\boldsymbol{B} \\ \boldsymbol{X} := \boldsymbol{Z}\boldsymbol{Z}\boldsymbol{v} \end{cases}$$

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
```

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
```

$\times$

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

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

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

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

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

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M}_2 \end{bmatrix} \quad \rightarrow ? \quad \begin{cases} \boldsymbol{M}_1 \boldsymbol{x}_T = \boldsymbol{y}_T \\ \boldsymbol{M}_2 \boldsymbol{x}_B = \boldsymbol{y}_B \end{cases}, \quad \begin{cases} \boldsymbol{y}_T := \boldsymbol{M}_1 \boldsymbol{x}_T \\ \boldsymbol{y}_B := \boldsymbol{M}_2 \boldsymbol{x}_B \end{cases} \qquad \times, \times$$

- $\mathrm{diag}(\boldsymbol{A} + \boldsymbol{B})$ vs. $\mathrm{diag}(\boldsymbol{A}) + \mathrm{diag}(\boldsymbol{B})$

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

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M}_1 & \mathbf{0} \\ \mathbf{0} & \boldsymbol{M}_2 \end{bmatrix} \quad \rightarrow ? \quad \begin{cases} \boldsymbol{M}_1 \boldsymbol{x}_T = \boldsymbol{y}_T \\ \boldsymbol{M}_2 \boldsymbol{x}_B = \boldsymbol{y}_B \end{cases}, \quad \begin{cases} \boldsymbol{y}_T := \boldsymbol{M}_1 \boldsymbol{x}_T \\ \boldsymbol{y}_B := \boldsymbol{M}_2 \boldsymbol{x}_B \end{cases} \quad \times, \times$$

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

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

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M}_2 \end{bmatrix} \quad \rightarrow ? \quad \begin{cases} \boldsymbol{M}_1 \boldsymbol{x}_T = \boldsymbol{y}_T \\ \boldsymbol{M}_2 \boldsymbol{x}_B = \boldsymbol{y}_B \end{cases}, \quad \begin{cases} \boldsymbol{y}_T := \boldsymbol{M}_1 \boldsymbol{x}_T \\ \boldsymbol{y}_B := \boldsymbol{M}_2 \boldsymbol{x}_B \end{cases} \quad \times, \times$$

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

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

Summary

- LAMPs are challenging — the optimal solution requires expertise in LA and HPC
[arXiv:1911.09421]

Summary

- LAMPs are challenging — the optimal solution requires expertise in LA and HPC
  [arXiv:1911.09421]

- To users: Beware!
  Compilers & languages are great with scalars, not so much with matrices (yet?)

Summary

- LAMPs are challenging — the optimal solution requires expertise in LA and HPC
  `[arXiv:1911.09421]`
- To users: Beware!
  Compilers & languages are great with scalars, not so much with matrices (yet?)
- To language developers: Please pay attention to the optimizations we exposed.
  They arise frequently in the solution of LAMPs.

Summary

- LAMPs are challenging — the optimal solution requires expertise in LA and HPC
  [arXiv:1911.09421]

- To users: Beware!
  Compilers & languages are great with scalars, not so much with matrices (yet?)

- To language developers: Please pay attention to the optimizations we exposed.
  They arise frequently in the solution of LAMPs.

- Our approach:
  "Linnea: Automatic Generation of Efficient Linear Algebra Programs", ACM TOMS, 2021
  `https://linnea.cs.umu.se/` [arXiv:1912.12924]

Summary

- LAMPs are challenging — the optimal solution requires expertise in LA and HPC
  `[arXiv:1911.09421]`

- To users: Beware!
  Compilers & languages are great with scalars, not so much with matrices (yet?)

- To language developers: Please pay attention to the optimizations we exposed.
  They arise frequently in the solution of LAMPs.

- Our approach:
  "Linnea: Automatic Generation of Efficient Linear Algebra Programs", ACM TOMS, 2021
  `https://linnea.cs.umu.se/` `[arXiv:1912.12924]`

Thank you for your attention!