High-performance matrix computations: It's not all about libraries

Paolo Bientinesi
pauldj@cs.umu.se

May 19, 2022
RWTH Aachen University

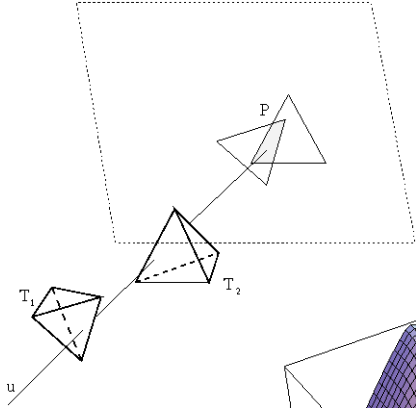
UMEÅ UNIVERSITY

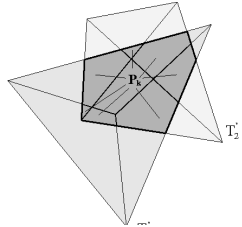
About me

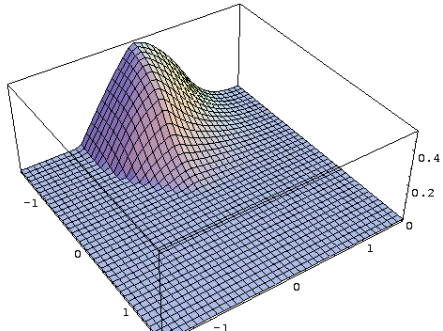
Italy – Tuscany

Italy – University of Pisa

Computational Geometry







USA – UT Austin, TX

- Symmetric eigenproblem
  $\boldsymbol{AX = X\Lambda}$
- FLAME project
  Automatic generation of algorithms

Algorithm $LU = A$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right), L \rightarrow \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right), U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right)$

where $A_{TL}, L_{TL}, U_{TL},$ are 0×0

While $m(A_{TL}) \leq m(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right),$$

$$\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} U_{00} & u_{01} & U_{02} \\ \hline 0 & \upsilon_{11} & u_{12}^T \\ \hline 0 & 0 & U_{22} \end{array}\right)$$

where $\alpha_{11}, 1, \upsilon_{11}$ are scalars

$$\upsilon_{11} := \alpha_{11} - l_{10}^T u_{01}$$
$$u_{12}^T := a_{12}^T - l_{10}^T U_{02}$$
$$l_{21} := (a_{21} - L_{20} u_{01}) / \upsilon_{11}$$

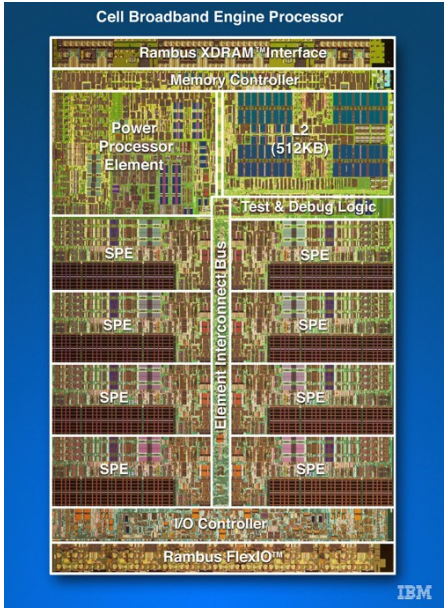
Continue with

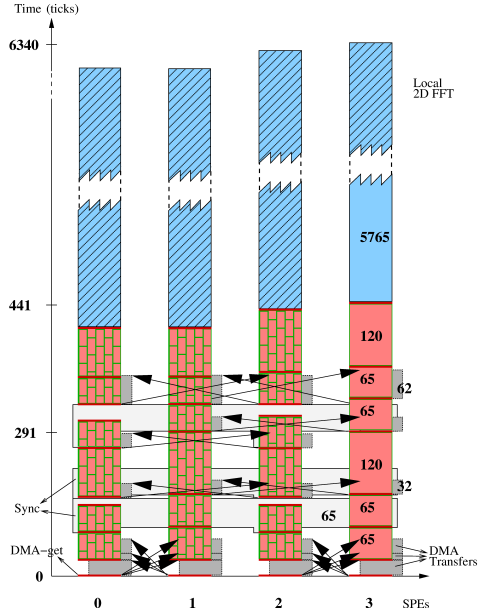
$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right), \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & 1 & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right), \cdots$$

endwhile

USA – Duke, NC

- Cell
- FFTs





Germany – RWTH Aachen University

High-Performance &
Automatic Computing

Modern
Inverse
Problems

Sweden – Umeå University

- Matrix & tensor operations
- HPC
- Computer music

HPC2N

High-Performance Computing Center North

Outline

- Part 1: Short intro
- Part 2: Numerical linear algebra: historical overview
- Part 3: Linear Algebra Mapping Problem (LAMP)
  Use and misuse of libraries

30-second lecture

30-second lecture

- Computers are great with numbers (scalars)

30-second lecture

- Computers are great with numbers (scalars)
- Domain experts work with vectors, matrices, and tensors

30-second lecture

- Computers are great with numbers (scalars)
- Domain experts work with vectors, matrices, and tensors
- Computers are not (yet?) great with those

30-second lecture

- Computers are great with numbers (scalars)
- Domain experts work with vectors, matrices, and tensors
- Computers are not (yet?) great with those
- Lots of (very good) libraries for specific operations exist

30-second lecture

- Computers are great with numbers (scalars)
- Domain experts work with vectors, matrices, and tensors
- Computers are not (yet?) great with those
- Lots of (very good) libraries for specific operations exist
- Is that enough? Sometimes

Libraries: back to the '70s!

- 1971: NAG Fortran Library – scientific computing
- 1972: EISPACK – eigenproblems
- 1973: BLAS 1 – basic vector ops
- 1975: LINPACK – linear systems
- ...
- ...

Approach

1. Identify a bottleneck
   e.g., linear system, eigenproblem, least-squares problem, SVD, ...

2. Encapsulate into a function
   e.g., `dgesv`, `dsyevd`, `dgelsx`, `dgesvd`, ...

3. Optimize, specialize

Basic linear algebra operations – 1973

"[..] a fairly small number of basic operations which are generally **responsible for a significant percentage** *of the total execution time"* – Hanson, Krogh, Lawson

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

1973 – 1979

- 1973: *"A proposal for standard linear algebra subprograms"* – Hanson, Krogh, Lawson
  Class I: DOT, ELVOP, G2, MG2 – Assembly
  Class II: NRM, XDOT, COPY, SWAP, SCALE, SUM, MAX – Fortran
- 1974: *"Standardization of FORTRAN callable subprograms for basic linear algebra"* – Lawson
- 1975–: LINPACK
- 1977: *"Basic Linear Algebra Subprograms for FORTRAN usage—an extended report"* – Hanson, Krogh, Kinkaid, Lawson
- 1977: *"Fortran BLAS timing"* – Dongarra
  Tests on 24 different computers

1979: BLAS 1

"Basic Linear Algebra Subprograms for FORTRAN usage"
— Hanson, Krogh, Kinkaid, Lawson (ACM TOMS)

"38 subprograms for basic operations of linear algebra"

- "aid in **design** and **coding** stages"
- "self-**documenting** quality of code"
- "a reduction of the execution time spent in these operations might be reflected in **cost savings** in the running of programs"
- "the programming of some of these low level operations involves **algorithmic and implementation subtleties** that are likely to be ignored"

Libraries

1970s

- Identification, analysis, optimization of **building blocks**

Libraries

1970s

- Identification, analysis, optimization of **building blocks**
- Computers difficult to program
  BUT
  Programs “easy” to optimize
- Cost($\mathcal{Alg}$) $\equiv$ #operations($\mathcal{Alg}$)

Libraries

1970s

- Identification, analysis, optimization of **building blocks**
- Computers difficult to program
  BUT
  Programs "easy" to optimize
- Cost($\mathcal{Alg}$) $\equiv$ #operations($\mathcal{Alg}$)
- **Libraries**: convenience, portability, separation of concerns, confidence performance

Libraries

1970s

- Identification, analysis, optimization of **building blocks**
- Computers difficult to program
  BUT
  Programs "easy" to optimize
- Cost($\mathcal{Alg}$) $\equiv$ #operations($\mathcal{Alg}$)
- **Libraries**: convenience, portability, separation of concerns, confidence performance

since 1980s

- Increasingly complex HW:
  E.g., vector processors, memory hierarchies, prefetching, ...

Libraries

1970s

- Identification, analysis, optimization of **building blocks**
- Computers difficult to program
  BUT
  Programs "easy" to optimize
- Cost($\mathcal{Alg}$) $\equiv$ #operations($\mathcal{Alg}$)
- **Libraries**: convenience, portability, separation of concerns, confidence performance

since 1980s

- Increasingly complex HW:
  E.g., vector processors, memory hierarchies, prefetching, ...
- Computers easier to program
  BUT
  Programs difficult to optimize
- Cost($\mathcal{Alg}$) $\not\equiv$ #operations($\mathcal{Alg}$)

Libraries

1970s

- Identification, analysis, optimization of **building blocks**
- Computers difficult to program
  BUT
  Programs "easy" to optimize
- Cost($\mathcal{Alg}$) $\equiv$ #operations($\mathcal{Alg}$)
- **Libraries**: convenience, portability, separation of concerns, confidence performance

since 1980s

- Increasingly complex HW:
  E.g., vector processors, memory hierarchies, prefetching, ...
- Computers easier to program
  BUT
  Programs difficult to optimize
- Cost($\mathcal{Alg}$) $\not\equiv$ #operations($\mathcal{Alg}$)
- **Libraries**: as before +
  necessity for performance

- 1988: BLAS 2 *"with some modern machine architectures, the use of the BLAS is not the best way to improve the efficiency of higher level codes. [..] the use of BLAS inhibits this optimization."*

  Matrix-vector operations **NOT** built on top of BLAS 1

- 1990: BLAS 3 *"Unfortunately, [BLAS 2] is often not well suited to computers with a hierarchy of memory"*

  Matrix-matrix operations **NOT** built on top of BLAS 1 & 2

- Immediate, widespread adoption: LAPACK, ScaLAPACK, PETSc, PLAPACK, ...

- Specialization, optimization, auto-tuning, high-level notation, automation, ...

Linear Algebra Libraries: 1970s

"Basic Linear Algebra Subprograms for FORTRAN usage", ACM TOMS, 1979

BLAS-1

Linear Algebra Libraries: 1980s

BLAS-2: Mat-vec ops, ACM TOMS 1988. BLAS-3: mat-mat ops, ACM TOMS 1990

BLAS-1, BLAS-2, BLAS-3

Linear Algebra Libraries: 1990s

Solvers & eigensolvers, 1992

LAPACK

BLAS-1, BLAS-2, BLAS-3

Linear Algebra Libraries: 1990s

Distributed Memory, 1995, 1997

ScaLAPACK, PLAPACK, ...

LAPACK

BLAS-1, BLAS-2, BLAS-3

Linear Algebra Libraries: 1990s

Dense & Sparse, 1997

PETSc, ...

ScaLAPACK, PLAPACK, ...

LAPACK

BLAS-1, BLAS-2, BLAS-3

Linear Algebra Libraries

and more!

PETSc, Trilinos, ...

ScaLAPACK, PLAPACK, Elemental, ...

LAPACK, Plasma, SuperMatrix, Magma, ...

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

Lin.alg. software landscape

Salient features

- Mature, expansive
- Community effort
- Standardized interface
- Careful organization: support routines, linear-systems, eigen-decompositions
- Clear layering: functionality, parallelism
- Well documented
- Performance-driven & HW-driven development

ACM Turing Award

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

Approach - ?

1. Identify the bottleneck
   e.g., linear system, eigenproblem, least-squares problem, SVD, ...

2. Encapsulate into a function
   e.g., `dgesv`, `dsyevd`, `dgelsx`, `dgesvd`, ...

3. Optimize

As opposed to what?

Approach - ?

1. Identify the bottleneck
   e.g., linear system, eigenproblem, least-squares problem, SVD, ...

2. Encapsulate into a function
   e.g., `dgesv`, `dsyevd`, `dgelsx`, `dgesvd`, ...

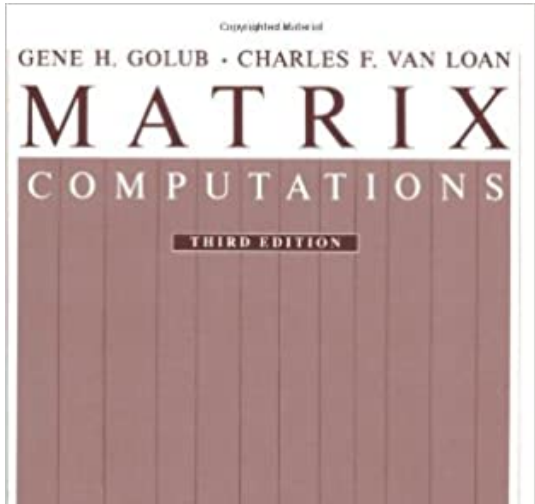
3. Optimize

As opposed to what? Workflow! Iteration vs. loop as a whole.

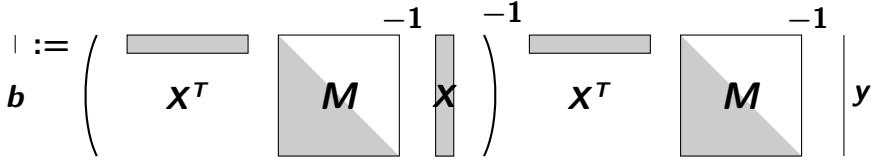
D. Fabregat, P. Bientinesi; ACM TOMS, 2014 [arXiv:1403.6426v1]
"Computing Petaflops over Terabytes of Data: The Case of Genome-Wide Association Studies"

C. Psarras, L. Karlsson, R. Bro, P. Bientinesi; ACM TOMS, 2022 [arXiv:2010.04678v2]
"Concurrent Alternating Least Squares for multiple simultaneous Canonical Polyadic Decompositions"

There is a matrix at your door



$$b := \left(X^T M^{-1} X\right)^{-1} X^T M^{-1} y$$



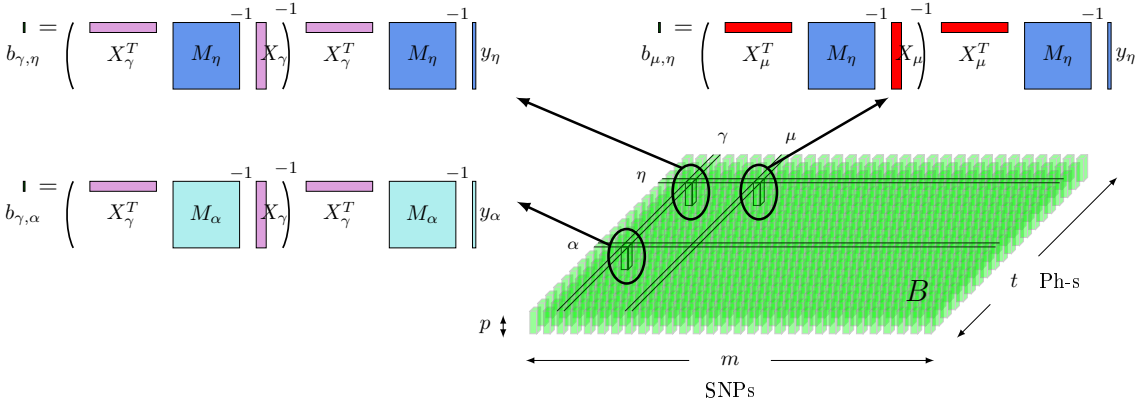
- $X \in \mathcal{R}^{n \times p}$ "SNP"
- $y \in \mathcal{R}^{n}$ "trait"
- $b \in \mathcal{R}^{p}$ "genetic effect"
- $M \in \mathcal{R}^{n \times n}$ "covariance matrix"

- $n \approx 1{,}000 - 50{,}000$
- $p \in [1, ..., 20]$
- M: SPD

"To be repeated millions of times"

Problem as a whole

$$\boldsymbol{b_{ij} := \left(X_i^T M_j^{-1} X_i\right)^{-1} X_i^T M_j^{-1} y_j}$$



Matrix computations

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
Ensemble Kalman Filter	$\delta X := \left(B^{-1} + H^TR^{-1}H\right)^{-1}H^TR^{-1}\left(Y - HX^b\right)$	
Ensemble Kalman Filter	$\delta X := XV^T\left(R + HX(HX)^T\right)^{-1}\left(Y - HX^b\right)$	
Image Restoration	$x_k := (H^TH + \lambda\sigma^2I_n)^{-1}(H^Ty + \lambda\sigma^2(v_{k-1} - u_{k-1}))$	
Image Restoration	$H^\dagger := H^T(HH^T)^{-1};\ y_k := H^\dagger y + (I_n - H^\dagger H)x_k$	
Rand. Matrix Inversion	$X_{k+1} := S(S^TAS)^{-1}S^T + (I_n - S(S^TAS)^{-1}S^TA)X_k(I_n - AS(S^TAS)^{-1}S^T)$	
Rand. Matrix Inversion	$X_{k+1} := X_k + WA^TS(S^TAWA^TS)^{-1}S^T(I_n - AX_k)$	$W \in \mathbb{R}^{n\times n}$, SPD
Rand. Matrix Inversion	$X_{k+1} := X_k + (I_n - X_kA^T)S(S^TA^TWAS)^{-1}S^TA^TW$	
Rand. Matrix Inversion	$\Lambda := S(S^TAWAS)^{-1}S^T;\ \Theta := \Lambda AW;\ M_k := X_kA - I$ $X_{k+1} := X_k - M_k\Theta - (M_k\Theta)^T + \Theta^T(AX_kA - A)\Theta$	

Matrix computations (2)

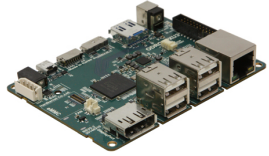
Generalized Least Squares	$b := (X^T M^{-1} X)^{-1} X^T M^{-1} y$	$n > m$; $M \in \mathbb{R}^{n \times n}$, SPD; $X \in \mathbb{R}^{n \times m}$; $y \in \mathbb{R}^{n \times 1}$
Stochastic Newton	$B_k := \frac{k}{k-1} B_{k-1}(I_n - A^T W_k((k-1)I_l + W_k^T A B_{k-1} A^T W_k)^{-1} W_k^T A B_{k-1})$	
Optimization	$x_f := WA^T(AWA^T)^{-1}(b - Ax)$; $x_o := W(A^T(AWA^T)^{-1}Ax - c)$	
Optimization	$x := W(A^T(AWA^T)^{-1}b - c)$	
Triangular Matrix Inv.	$X_{10} := L_{10}L_{00}^{-1}$; $X_{20} := L_{20} + L_{22}^{-1}L_{21}L_{11}^{-1}L_{10}$; $X_{11} := L_{11}^{-1}$; $X_{21} := -L_{22}^{-1}L_{21}$	
Tikhonov Regularization	$x := (A^T A + \Gamma^T \Gamma)^{-1} A^T b$	$A \in \mathbb{R}^{n \times m}$; $\Gamma \in \mathbb{R}^{m \times m}$; $b \in \mathbb{R}^{n \times 1}$
Tikhonov Regularization	$x := (A^T A + \alpha^2 I)^{-1} A^T b$	
Gen. Tikhonov Reg.	$x := (A^T PA + Q)^{-1}(A^T Pb + Qx_0)$	$P \in \mathbb{R}^{n \times n}$, SSPD; $Q \in \mathbb{R}^{m \times m}$, SSPD; $x_0 \in \mathbb{R}^{m \times 1}$
Gen. Tikhonov reg.	$x := x_0 + (A^T PA + Q)^{-1}(A^T P(b - Ax_0))$	
LMMSE estimator	$K_{t+1} := C_t A^T (AC_t A^T + C_z)^{-1}$; $x_{t+1} := x_t + K_{t+1}(y - Ax_t)$; $C_{t+1} := (I - K_{t+1}A)C_t$	
LMMSE estimator	$x_{\text{out}} = C_X A^T (AC_X A^T + C_Z)^{-1}(y - Ax) + x$	
LMMSE estimator	$x_{\text{out}} := (A^T C_Z^{-1} A + C_X^{-1})^{-1} A^T C_Z^{-1}(y - Ax) + x$	

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

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

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

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



MUL | ADD | MOV

MOVAPD

VFMADDPD ...

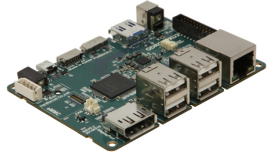
$$K_k := P_k^b H^T (H P_k^b H^T + R)^{-1}; \quad x_k^a := x_k^b + K_k(z_k - H x_k^b); \quad P_k^a := (I - K_K H)\, P_k^b$$

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

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

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

?



MUL | ADD | MOV

MOVAPD

VFMADDPD ...

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

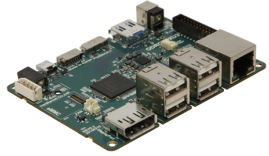
$$\begin{cases} C_\dagger := PCP^T + Q \\ K := C_\dagger H^T (H C_\dagger H^T)^{-1} \end{cases}$$

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

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

?

2-step solution



MUL | ADD | MOV

MOVAPD

VFMADDPD ...

Step 1: Libraries of building blocks

Support for all kinds of matrix types, parallelism, and platforms

Step 1: Libraries of building blocks

Support for all kinds of matrix types, parallelism, and platforms

PETSc, Trilinos, ...

ScaLAPACK, PLAPACK, Elemental, ...

LAPACK, Plasma, SuperMatrix, Magma, ...

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

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

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

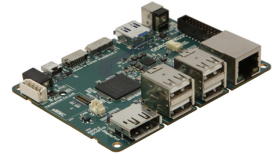
$$\Lambda := S(S^T AWAS)^{-1} S^T; \ \Theta := \Lambda AW; \ M_k := X_k A - I$$
$$X_{k+1} := X_k - M_k\Theta - (M_k\Theta)^T + \Theta^T(AX_kA - A)\Theta$$

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

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

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

... **BLAS** **LAPACK** ...



MUL ADD MOV

MOVAPD

VFMADDPD ...

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

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

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

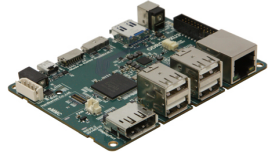
$$x := A(B^TB + A^TR^T\Lambda RA)^{-1}B^TBA^{-1}y \quad \ldots \quad E := Q^{-1}U(I + U^TQ^{-1}U)^{-1}U^T$$

?

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

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

... BLAS LAPACK ...



MUL | ADD | MOV

MOVAPD

VFMADDPD ...

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

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

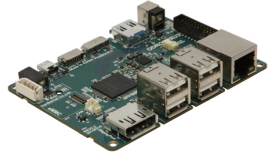
$$\Lambda := S(S^T A W A S)^{-1} S^T; \ \Theta := \Lambda A W; \ M_k := X_k A - I$$
$$X_{k+1} := X_k - M_k \Theta - (M_k \Theta)^T + \Theta^T (A X_k A - A)\Theta$$

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

Step 2:
LINEAR ALGEBRA MAPPING PROBLEM ("LAMP")

$$y := \alpha x + y \qquad LU = A \qquad \cdots \qquad C := \alpha AB + \beta C$$
$$X := A^{-1} B \qquad C := AB^T + BA^T + C \qquad X := L^{-1} M L^{-T} \qquad QR = A$$

... BLAS LAPACK ...



MUL ADD MOV
MOVAPD
VFMADDPD ...

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

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

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

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

Step 2:
LINEAR ALGEBRA MAPPING PROBLEM ("LAMP")

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

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

... BLAS LAPACK ...

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

A. Sankaran, N.A. Alashti, C. Psarras, P. Bientinesi, [arXiv:2202.09888]
"Benchmarking the Linear Algebra Awareness of TensorFlow and PyTorch", iWAPT-22

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

Linear Algebra Mapping Problem

Linear Algebra Mapping Problem

- $\mathcal{E}$: a sequence of explicit assignments $\boldsymbol{var_i := EXP_i}$

Linear Algebra Mapping Problem

- $\mathcal{E}$: a sequence of explicit assignments $\quad \boldsymbol{var_i := EXP_i}$
- $\mathcal{K}$: a set of available computational building blocks $\quad$ e.g., BLAS, LAPACK, ...

Linear Algebra Mapping Problem

- $\mathcal{E}$: 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

Linear Algebra Mapping Problem

- $\mathcal{E}$: 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 $\mathcal{E}$.

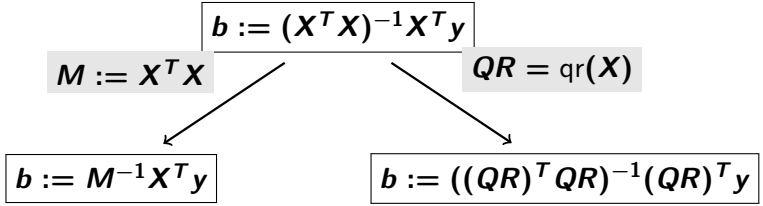
Linear Algebra Mapping Problem

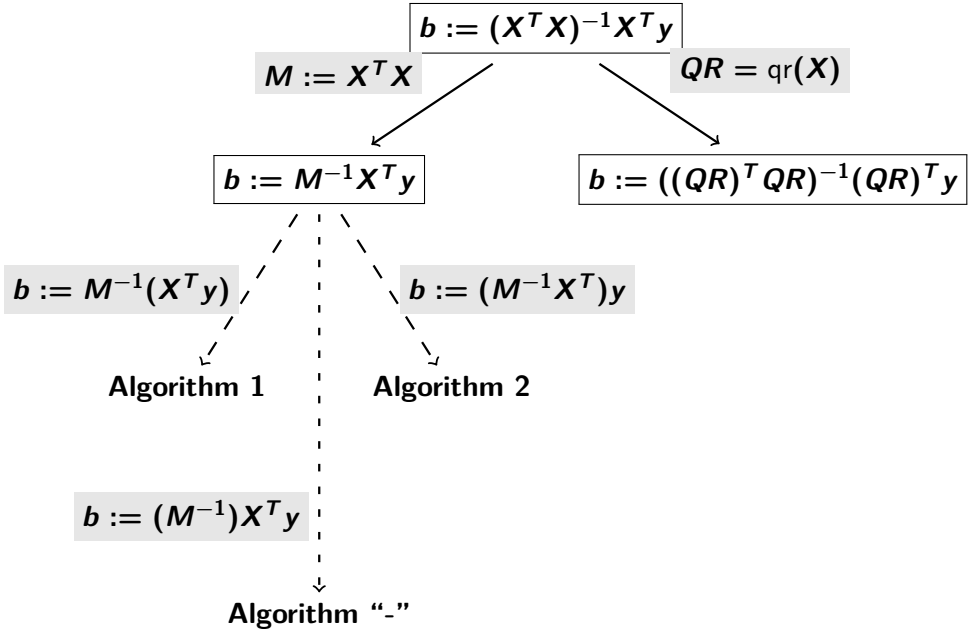
- $\mathcal{E}$: a sequence of explicit assignments — $\boldsymbol{var_i := EXP_i}$
- $\mathcal{K}$: a set of available computational building blocks — e.g., BLAS, LAPACK, ...
- $\mathcal{M}$: a cost function defined over $\mathcal{K}^+$ — #FLOPs, exec. time, #mem.ops, stability

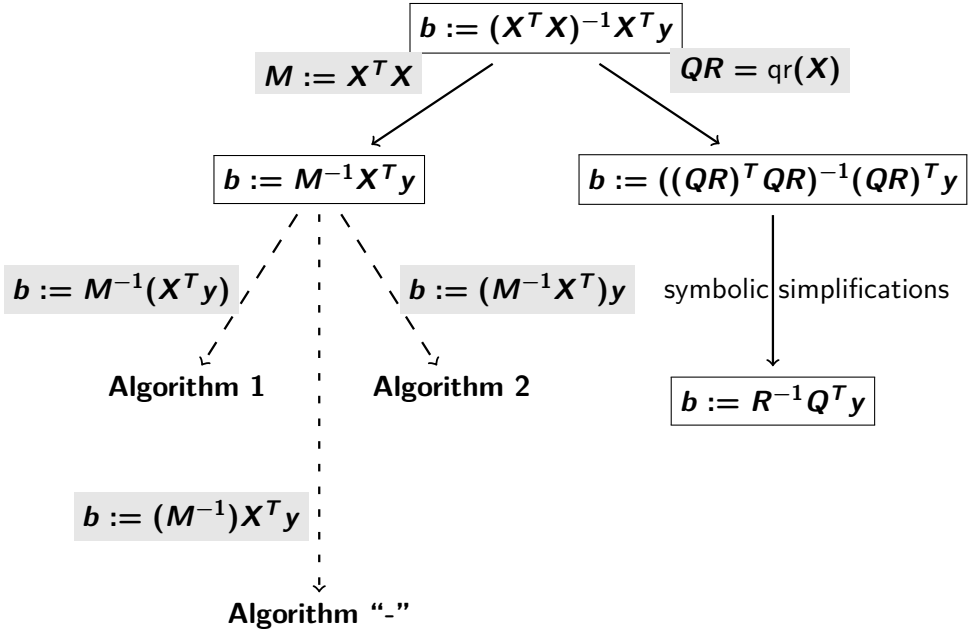
LAMP:
Find a sequence of calls to building blocks in $\mathcal{K}$, optimal according to $\mathcal{M}$, that computes all the assignments in $\mathcal{E}$.

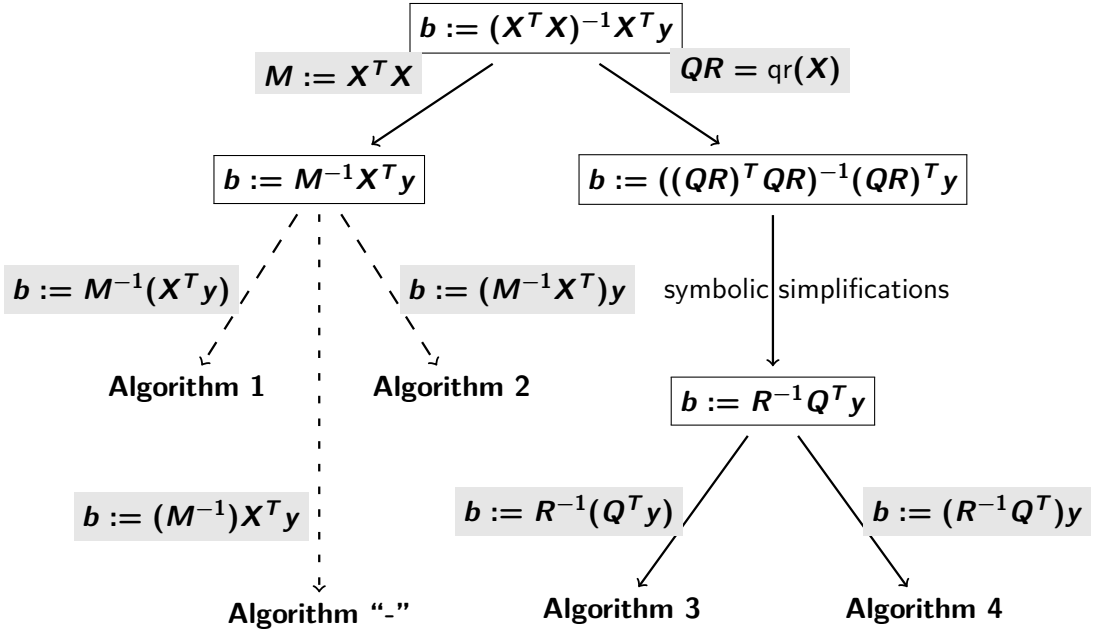
- Suboptimal solution $\longrightarrow$ easy
- Optimality $\longrightarrow$ NP complete $\longleftarrow$ reduction from Ensemble Computation

$$\boxed{b := (X^T X)^{-1} X^T y}$$


$b := (X^T X)^{-1} X^T y$
$M := X^T X$
$QR = \text{qr}(X)$
$b := M^{-1} X^T y$
$b := ((QR)^T QR)^{-1} (QR)^T y$


$b := (X^T X)^{-1} X^T y$
$M := X^T X$
$QR = \mathrm{qr}(X)$
$b := M^{-1} X^T y$
$b := ((QR)^T QR)^{-1} (QR)^T y$
$b := M^{-1}(X^T y)$
$b := (M^{-1} X^T) y$
Algorithm 1
Algorithm 2
$b := (M^{-1}) X^T y$
Algorithm "-"


$b := (X^T X)^{-1} X^T y$
$M := X^T X$
$QR = \mathrm{qr}(X)$
$b := M^{-1} X^T y$
$b := ((QR)^T QR)^{-1} (QR)^T y$
$b := M^{-1}(X^T y)$
$b := (M^{-1} X^T) y$
symbolic simplifications
Algorithm 1
Algorithm 2
$b := R^{-1} Q^T y$
$b := (M^{-1}) X^T y$
Algorithm "-"


$b := (X^T X)^{-1} X^T y$
$M := X^T X$
$QR = \mathrm{qr}(X)$
$b := M^{-1} X^T y$
$b := ((QR)^T QR)^{-1} (QR)^T y$
$b := M^{-1}(X^T y)$
$b := (M^{-1} X^T) y$
symbolic simplifications
Algorithm 1
Algorithm 2
$b := R^{-1} Q^T y$
$b := (M^{-1}) X^T y$
$b := R^{-1}(Q^T y)$
$b := (R^{-1} Q^T) y$
Algorithm "-"
Algorithm 3
Algorithm 4

Problem acknowledged, yet overlooked

"How do users go about solving their matrix computations?"

Problem acknowledged, yet overlooked

"How do users go about solving their matrix computations?"

1) By hand! C/Fortran Required: expertize, time, patience

computer efficiency!  ... but human productivity 

Problem acknowledged, yet overlooked

"How do users go about solving their matrix computations?"

1) By hand! C/Fortran Required: expertize, time, patience

computer efficiency!  . . . but human productivity 

Fast, smart solutions — painstaking development

Problem acknowledged, yet overlooked

"How do users go about solving their matrix computations?"

1) By hand! C/Fortran Required: expertize, time, patience

computer efficiency!  ... but human productivity 

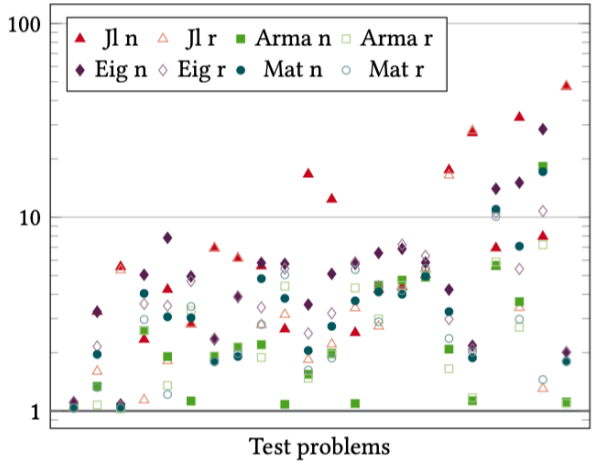
Fast, smart solutions — painstaking development

2) High-level languages: Matlab, Julia, R, Armadillo (C++), NumPy, TensorFlow, ...

human productivity!  ... computer efficiency?

Computer efficiency?

Languages "slowdowns" wrt fast solutions – sequential execution



Jl: Julia, **Arma**: Armadillo, **Eig**: Eigen, **Mat**: Matlab. **n**/**r**: naive/recommended implementation

Problem acknowledged, yet overlooked

"How do users go about solving their matrix computations?"

1) By hand! C/Fortran Required: expertize, time, patience

computer efficiency!  . . . but human productivity 

Problem acknowledged, yet overlooked

"How do users go about solving their matrix computations?"

1) By hand! C/Fortran Required: expertize, time, patience

computer efficiency!  . . . but human productivity 

Fast, smart solutions — painstaking development

Problem acknowledged, yet overlooked

"How do users go about solving their matrix computations?"

1) By hand! C/Fortran Required: expertize, time, patience

computer efficiency!  ... but human productivity 

Fast, smart solutions — painstaking development

2) High-level languages: Matlab, Julia, R, Armadillo (C++), NumPy, TensorFlow, ...

human productivity!  ... but computer efficiency 

Problem acknowledged, yet overlooked

"How do users go about solving their matrix computations?"

1) By hand! C/Fortran Required: expertize, time, patience

computer efficiency!  ... but human productivity 

Fast, smart solutions — painstaking development

2) High-level languages: Matlab, Julia, R, Armadillo (C++), NumPy, TensorFlow, ...

human productivity!  ... but computer efficiency 

Speedy development — slow solutions $\longrightarrow$ "2-language problem"

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

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

Seven languages/libraries. Not an exhaustive survey.

Main criterion: high level of **abstraction**. Also: popularity, expressiveness, (performance...)

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

Seven languages/libraries. Not an exhaustive survey.

Main criterion: high level of **abstraction**. Also: popularity, expressiveness, (performance...)

```
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
```

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

Seven languages/libraries. Not an exhaustive survey.

Main criterion: high level of **abstraction**. Also: popularity, expressiveness, (performance...)

```
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
```

Investigation then adapted and extended to TensorFlow and PyTorch

Do they map? matrix products

	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
$C = AB$								

Do they map? matrix products

	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
$C = AB$	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27

Do they map? matrix products

	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
$C = AB$	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27
`GEMM`	✓	✓	✓	✓	✓	✓	✓	

Do they map? matrix products

	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
C = AB	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27
`GEMM`	✓	✓	✓	✓	✓	✓	✓	

C = C+AA'

Do they map? matrix products

	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
C = AB	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27
`GEMM`	✓	✓	✓	✓	✓	✓	✓	
C = C+AA'	0.18	0.17	0.21	0.32	0.29	0.17	0.18	0.14

Do they map? matrix products

	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
C = AB	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27
`GEMM`	✓	✓	✓	✓	✓	✓	✓	
C = C+AA'	0.18	0.17	0.21	0.32	0.29	0.17	0.18	0.14
`SYRK`	✓	✓	✓	×	×	✓	✓	

Do they map? matrix products

	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
C = AB	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27
`GEMM`	✓	✓	✓	✓	✓	✓	✓	
C = C+AA'	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

Do they map? matrix products

	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
C = AB	0.29	0.28	0.30	0.31	0.29	0.29	0.29	0.27
`GEMM`	✓	✓	✓	✓	✓	✓	✓	
C = C+AA'	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? $\boldsymbol{Ax = b} \quad \equiv \quad \boldsymbol{x := A \backslash b} \quad \not\equiv \quad \text{inv}(\boldsymbol{A}) * \boldsymbol{b}$

	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
$\boldsymbol{x := A \backslash b}$								

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

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

	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
$\text{inv}(\boldsymbol{A}) * \boldsymbol{b}$	1.76	1.82	1.69	2.20	2.21	**0.63**	2.49	1.71

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

	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
$\text{inv}(\boldsymbol{A}) * \boldsymbol{b}$	1.76	1.82	1.69	2.20	2.21	**0.63**	2.49	1.71
LinSolve	-	-	-	-	-	✓	-	-

but ...

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

	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
$\text{inv}(\boldsymbol{A}) * \boldsymbol{b}$	1.76	1.82	1.69	2.20	2.21	**0.63**	2.49	1.71
LinSolve	-	-	-	-	-	✓	-	-

but . . . should they map?

Remember

$\alpha, \beta, x \in \mathbf{R}, \quad A, B, C, X \in \mathbf{R^{m,n}}$

Commutativity($*$)

$$x := \alpha * \beta * \alpha^{-1} \quad \Rightarrow \quad x := \beta$$

$$X := A * B * A^{-1} \quad \not\Rightarrow \quad x := B$$

Remember

$\alpha, \beta, x \in \mathbf{R}, \quad \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{X} \in \mathbf{R}^{m,n}$

Commutativity($*$)

$$x := \alpha * \beta * \alpha^{-1} \quad \Rightarrow \quad x := \beta$$

$$\boldsymbol{X} := \boldsymbol{A} * \boldsymbol{B} * \boldsymbol{A}^{-1} \quad \not\Rightarrow \quad x := \boldsymbol{B}$$

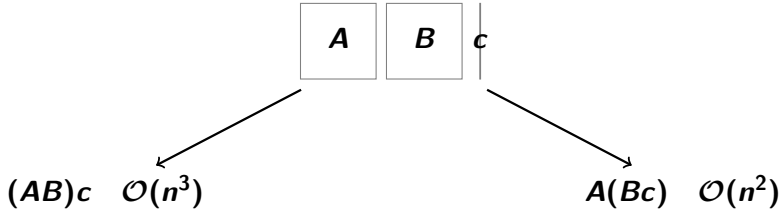
Associativity($*$)

$$\boldsymbol{X} := (\boldsymbol{A} * \boldsymbol{B}) * \boldsymbol{C} \quad \equiv \quad \boldsymbol{X} := \boldsymbol{A} * (\boldsymbol{B} * \boldsymbol{C})$$

But

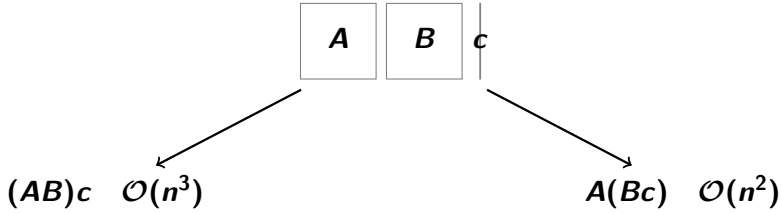
$$\text{Cost}\,((\boldsymbol{A} * \boldsymbol{B}) * \boldsymbol{C}) \quad \not\equiv \quad \text{Cost}\,(\boldsymbol{A} * (\boldsymbol{B} * \boldsymbol{C}))$$

Optimal parenthesisation



Matrix product is associative, but its cost is not

Optimal parenthesisation



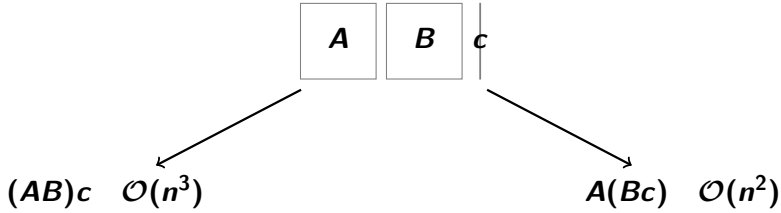
Matrix product is associative, but its cost is not

Matrix Chain(k)

Which is the best one?

Catalan$(k-1)$ parenthesiszations
if parallelism, then $k!$ parenthesiszations

Optimal parenthesisation



Matrix product is associative, but its cost is not

Matrix Chain(k)

Catalan$(k-1)$ parenthesiszations
if parallelism, then $k!$ parenthesiszations

Which is the best one?

Matrix Chain Algorithm(k)

$O(k \log k)$ Hu & Shing 1982
$O(k^3)$ dynamic programming

Matrix Chain?

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

Matrix Chain?

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

		Matlab	Octave	Julia	R	Eigen	Armad.	NumPy
1)	LtR no par.	0.056	0.056	0.055	0.061	0.058	0.056	0.055
	LtR guided	0.056	0.056	0.055	0.061	0.058	0.056	0.055

Matrix Chain?

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

		Matlab	Octave	Julia	R	Eigen	Armad.	NumPy
1)	LtR no par.	0.056	0.056	0.055	0.061	0.058	0.056	0.055
	LtR guided	0.056	0.056	0.055	0.061	0.058	0.056	0.055
2)	RtL no par.	0.42	0.43	0.42	0.44	0.42	**0.055**	0.42
	RtL guided	0.055	0.056	0.054	0.059	0.056	0.055	0.056

Matrix Chain?

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

		Matlab	Octave	Julia	R	Eigen	Armad.	NumPy
1)	LtR no par.	0.056	0.056	0.055	0.061	0.058	0.056	0.055
	LtR guided	0.056	0.056	0.055	0.061	0.058	0.056	0.055
2)	RtL no par.	0.42	0.43	0.42	0.44	0.42	**0.055**	0.42
	RtL guided	0.055	0.056	0.054	0.059	0.056	0.055	0.056
3)	Mix no par.	0.32	0.33	0.33	0.33	0.35	0.31	0.33
	Mix guided	0.21	0.22	0.22	0.22	0.23	0.20	0.22

Matrix Chain?

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

		Matlab	Octave	Julia	R	Eigen	Armad.	NumPy
1)	LtR no par.	0.056	0.056	0.055	0.061	0.058	0.056	0.055
	LtR guided	0.056	0.056	0.055	0.061	0.058	0.056	0.055
2)	RtL no par.	0.42	0.43	0.42	0.44	0.42	**0.055**	0.42
	RtL guided	0.055	0.056	0.054	0.059	0.056	0.055	0.056
3)	Mix no par.	0.32	0.33	0.33	0.33	0.35	0.31	0.33
	Mix guided	0.21	0.22	0.22	0.22	0.23	0.20	0.22
	Matrix chains	×	×	×	×	×	≈	×

Challenge: Generalized Matrix Chain

[arXiv:1804.04021]

Matrix chains 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., $\boldsymbol{A} \rightarrow \boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{Q}$, $\boldsymbol{A} \rightarrow \boldsymbol{L}\boldsymbol{U}$)
- Properties & specialized kernels (`GEMM`, `TRMM`, `SYMM`, ...)

Challenge: Generalized Matrix Chain

[arXiv:1804.04021]

Matrix chains 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., $\boldsymbol{A} \rightarrow \boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{Q}$, $\boldsymbol{A} \rightarrow \boldsymbol{L}\boldsymbol{U}$)
- Properties & specialized kernels (`GEMM`, `TRMM`, `SYMM`, ...)

$\Rightarrow$ **Generalized** Matrix Chain Algorithm

Challenge: Not all flops were created equal

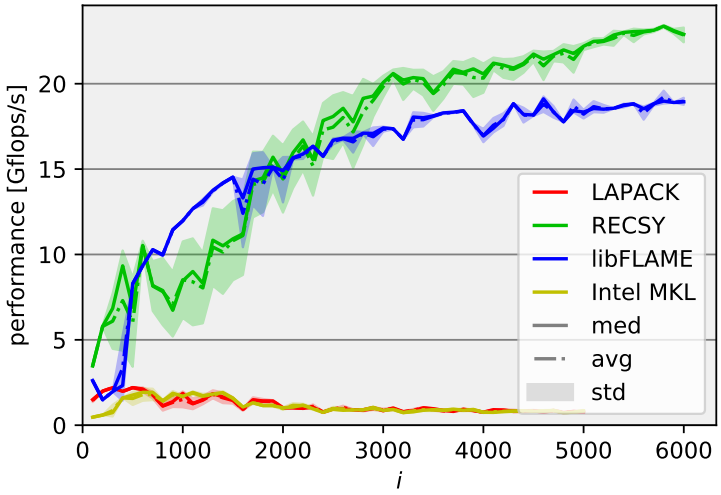
#FLOPs vs. execution time (vs. numerical stability)

$$\operatorname*{argmin}_{\mathcal{A}} \left(\text{FLOPs}(\mathcal{A}) \right) \neq \operatorname*{argmin}_{\mathcal{A}} \left(\text{time}(\mathcal{A}) \right)$$

Challenge: Not all flops were created equal

[arXiv:1504.08035]

Triangular Sylvester equation – all libraries perform the same # of flops



Challenge: Not all flops were created equal

[arXiv:1706.01341]

#FLOPs vs. execution time (vs. numerical stability)

$$\operatorname*{\mathbf{argmin}}_{\mathcal{A}} \left(\text{FLOPs}(\mathcal{A}) \right) \neq \operatorname*{\mathbf{argmin}}_{\mathcal{A}} \left(\text{time}(\mathcal{A}) \right)$$

$\Rightarrow$ **Performance prediction**: efficiency

Challenge: Not all flops were created equal [arXiv:1706.01341]

#FLOPs vs. execution time (vs. numerical stability)

$$\underset{\mathcal{A}}{\textbf{argmin}}\ (\ \text{FLOPs}(\mathcal{A})\ ) \neq \underset{\mathcal{A}}{\textbf{argmin}}\ (\ \text{time}(\mathcal{A})\ )$$

$\Rightarrow$ **Performance prediction**: efficiency

Big Challenge: **Parallelism**

$$\boldsymbol{X} := \boldsymbol{A}((\boldsymbol{B}^T\boldsymbol{C}^{-T})\boldsymbol{D}) \quad \text{vs.} \quad \boldsymbol{X} := (\boldsymbol{A}\boldsymbol{B}^T)(\boldsymbol{C}^{-T}\boldsymbol{D}) \quad \text{vs.} \quad \ldots$$

Properties?

Operation	Property	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
Linear System	-	0.71	0.74	0.62	0.67	0.63	0.62	0.65	0.61

Properties?

Operation	Property	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
Linear System	-	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

Properties?

Operation	Property	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
Linear System	-	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

Properties?

Operation	Property	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
Linear System	-	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

Properties?

Operation	Property	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
Linear System	-	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
		$\approx$	$\approx$	$\approx$	$\times$	$\times$	$\approx$	$\times$	

Properties?

Operation	Property	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy	C
Linear System	-	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
		$\approx$	$\approx$	$\approx$	$\times$	$\times$	$\approx$	$\times$	
Multiplication	-	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
		$\times$	$\times$	✓	$\times$	$\times$	$\times$	$\times$	

Challenge: Inference of properties

- **easy** $\quad E := L_1 * U^T * L_2 \quad$ triangular(E) ?

Challenge: Inference of properties

- **easy** $\boldsymbol{E := L_1 * U^T * L_2}$ triangular($\boldsymbol{E}$) ?
- **hard** $\boldsymbol{\lambda(L^{-T} A L^{-1})}$ symmetric($\boldsymbol{L^{-T} A L^{-1}}$) ?

Challenge: Inference of properties

- **easy** $E := L_1 * U^T * L_2$ triangular(E) ?
- **hard** $\lambda(L^{-T} A L^{-1})$ symmetric($L^{-T} A L^{-1}$) ?
- **impossible?** $E := Q^{-1} U (I + U^T Q^{-1} U)^{-1} U^T$ properties($I + U^T Q^{-1} U$) ?

Challenge: Inference of properties

- **easy** $E := L_1 * U^T * L_2$ triangular(E) ?
- **hard** $\lambda(L^{-T}AL^{-1})$ symmetric($L^{-T}AL^{-1}$) ?
- **impossible?** $E := Q^{-1}U(I + U^TQ^{-1}U)^{-1}U^T$ properties($I + U^TQ^{-1}U$) ?

$\Rightarrow$ **Symbolic analysis**: pattern matching

Challenge: Common Subexpressions

[arXiv:1710.06915]

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

Challenge: Common Subexpressions

[arXiv:1710.06915]

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

BUT

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

Common Subexpressions?

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

Common Subexpressions?

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

	Matlab	Octave	Julia	R	Eigen	Armad.	NumPy
copy	0.27	0.31	0.36	0.30	0.30	0.26	0.30

Common Subexpressions?

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

	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
	×	×	×	×	×	×	×

Other features (challenges)

- Code motion

Other features (challenges)

- Code motion

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

Other features (challenges)

- Code motion

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

$\rightarrow$?

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

Other features (challenges)

- Code motion

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

$\rightarrow$?

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

$\times$

Other features (challenges)

- Code motion

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

$\rightarrow$?

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

$\times$

- Blocked operands

$$M = \begin{bmatrix} M_1 & 0 \\ 0 & M_2 \end{bmatrix}$$

Other features (challenges)

- Code motion

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

$\rightarrow$?

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

$\times$

- Blocked operands

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

Other features (challenges)

- Code motion

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

$\rightarrow$?

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

$\times$

- Blocked operands

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

Other features (challenges)

- Code motion

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

$\rightarrow$?

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

$\times$

- Blocked operands

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

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

Other features (challenges)

- Code motion

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

$\rightarrow$?

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

$\times$

- Blocked operands

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

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

Other features (challenges)

- Code motion

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

$\rightarrow$?

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

$\times$

- Blocked operands

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

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

  $\mathrm{diag}(\boldsymbol{AB})$ vs. ...

Other features (challenges)

- Code motion

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

$\rightarrow$?

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

$\times$

- Blocked operands

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

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

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

Summary

- This evaluation is **NOT** a ranking of languages [arXiv:1911.09421]

Summary

- This evaluation is **NOT** a ranking of languages [arXiv:1911.09421]
- We expose optimizations that arise frequently in the solution of LAMPs
- Compilers & languages are great with scalars, not so much with matrices

Summary

- This evaluation is **NOT** a ranking of languages [arXiv:1911.09421]
- We expose optimizations that arise frequently in the solution of LAMPs
- Compilers & languages are great with scalars, not so much with matrices
- LAMPs are challenging — the optimal solution requires lots of interdisciplinary expertise

Summary

- This evaluation is **NOT** a ranking of languages [arXiv:1911.09421]
- We expose optimizations that arise frequently in the solution of LAMPs
- Compilers & languages are great with scalars, not so much with matrices
- LAMPs are challenging — the optimal solution requires lots of interdisciplinary expertise
- Next: "**Linnea**: A linear algebra compiler" [arXiv:1912.12924]

Linnea

`https://linnea.cs.umu.se/`

Linnea

Linear algebra knowledge: operators, identities, properties, theorems

- Distributivity, commutativity, partitionings, ...
- Matrix chains, generalize matrix chains
- $((QR)^T QR)^{-1}(QR)^T y \;\rightarrow\; (R^T Q^T QR)^{-1} R^T Q^T y \;\rightarrow\; R^{-1} R^{-T} R^T Q^T y \;\rightarrow\; R^{-1} Q^T y$
- $\mathrm{SPD}(A) \rightarrow \mathrm{SPD}(A_{BR} - A_{BL} A_{TL}^{-1} A_{BL}^T)$ Schur complement
- ...

Linnea: Example

$$\boldsymbol{w} := \boldsymbol{A}\boldsymbol{B}^{-1}\boldsymbol{c}, \quad \text{SPD}(\boldsymbol{B})$$

Naive

```
w = A*inv(B)*c
```

Recommended

```
w = A*(B\c)
```

Expert

```
L = Chol(B)
w = A*(L'\(L\c))
```

Linnea: Example

$$\boldsymbol{w} := \boldsymbol{A}\boldsymbol{B}^{-1}\boldsymbol{c}, \quad \mathrm{SPD}(\boldsymbol{B})$$

Naive

```
w = A*inv(B)*c
```

Recommended

```
w = A*(B\c)
```

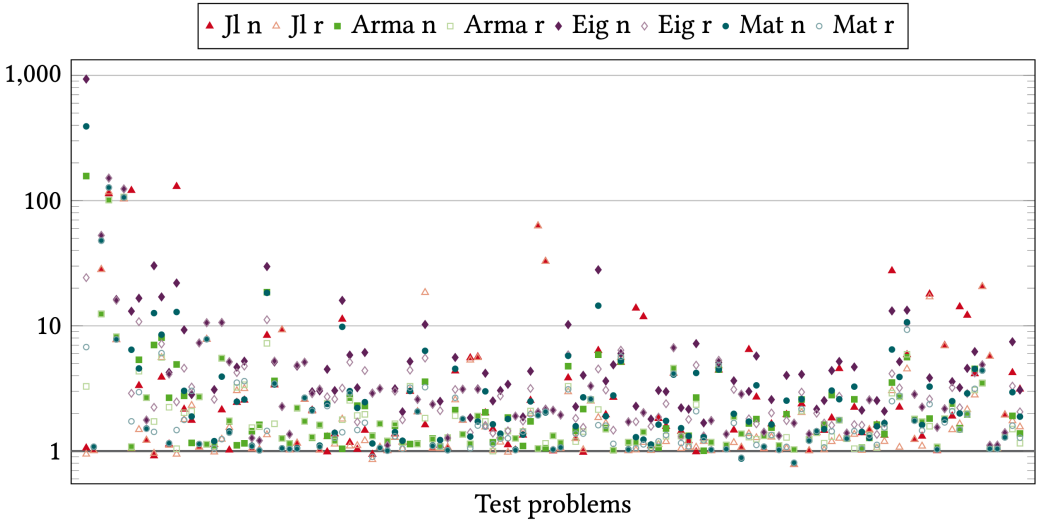
Expert

```
L = Chol(B)
w = A*(L'\(L\c))
```

Generated

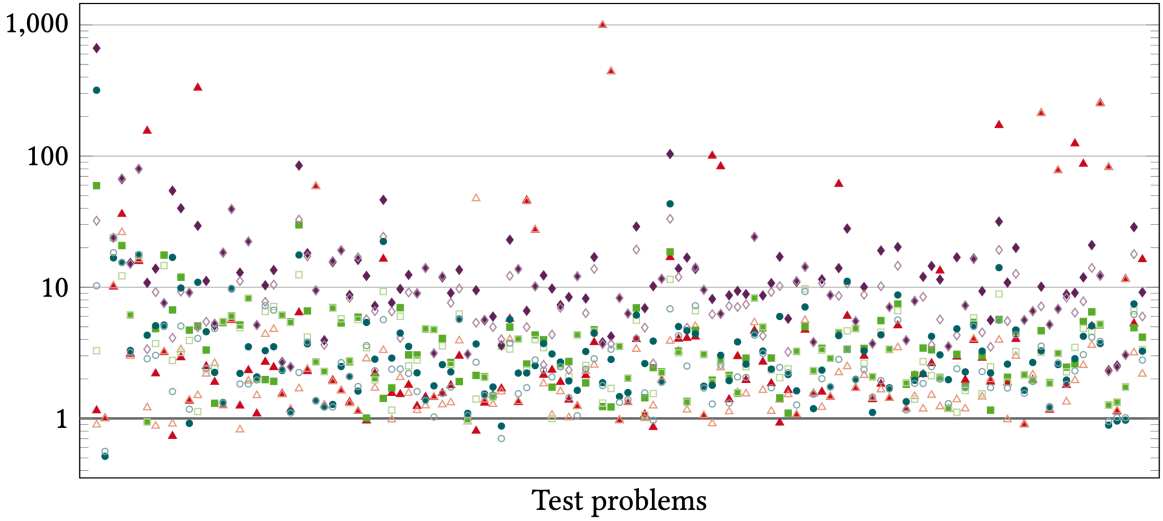
```
ml0 = A; ml1 = B; ml2 = c;
potrf!('L', ml1)
trsv!('L', 'N', 'N', ml1, ml2)
trsv!('L', 'T', 'N', ml1, ml2)
ml3 = Array{Float64}(10)
gemv!('N', 1.0, ml0, ml2, 0.0, ml3)
w = ml3
```

Linnea speedup: 1 thread



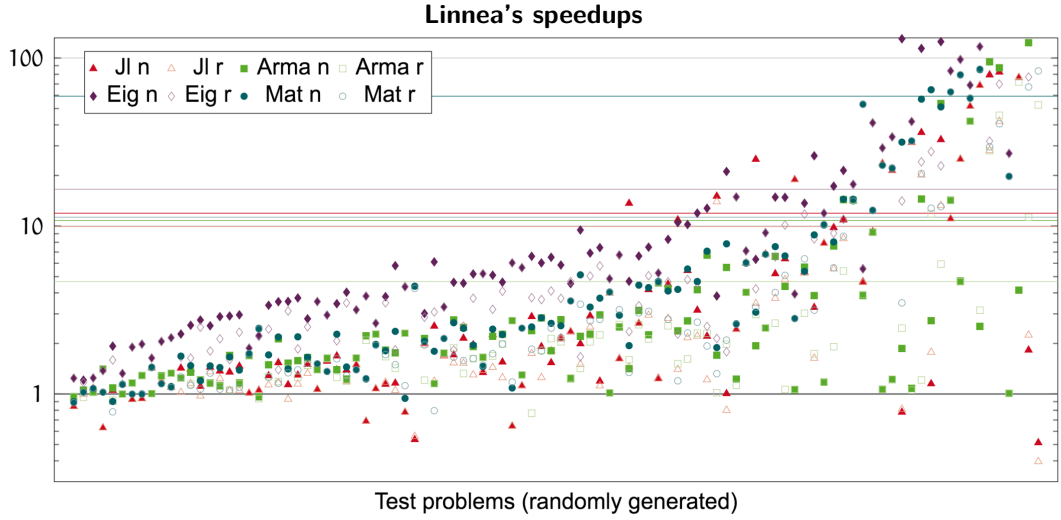
Jl: Julia, **Arma**: Armadillo, **Eig**: Eigen, **Mat**: Matlab. **n**/**r**: naive/recommended implementation

Linnea speedup: 24 threads



Jl: Julia, **Arma**: Armadillo, **Eig**: Eigen, **Mat**: Matlab. **n**/**r**: naive/recommended implementation

Linnea: Results for random expressions



Jl: Julia, **Arma**: Armadillo, **Eig**: Eigen, **Mat**: Matlab. **n**/**r**: naive/recommended implementation

What's next?

-
-
-

What's next?

-
-
-

THANK YOU!!

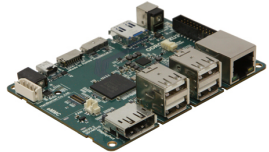
Tensor-land

Quantum Chemistry | High-order FEM | DG | ... | Tensor App #N

Transposition | Contraction | ... | Alternating LS

Khatri-Rao | SpMTTKRP | ... | TTV, TTM

HPTT **TCL** **...** **BLAS**



MUL | ADD | MOV

MOVAPD

VFMADDPD | ...