Knowledge-Based Automatic Generation of Algorithms and Code

Diego Fabregat Traver

AICES, RWTH Aachen
fabregat@aices.rwth-aachen.de

Doctoral Defense
Aachen, December 6th, 2013

Focus

Design and implementation of Domain-Specific Compilers for Linear Algebra matrix equations.

Focus

Design and implementation of Domain-Specific Compilers for Linear Algebra matrix equations.

Why?

- Matrix equations are ubiquitous

Introduction

Focus

Design and implementation of Domain-Specific Compilers for Linear Algebra matrix equations.

Why?

- Matrix equations are ubiquitous
- Complex and time consuming development

Focus

Design and implementation of Domain-Specific Compilers for Linear Algebra matrix equations.

Why?

- Matrix equations are ubiquitous
- Complex and time consuming development
- It requires expertise from multiple areas:
  - Application domain
  - Numerics, algorithmics
  - High-performance computing

Focus

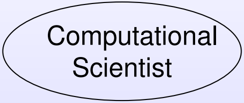
Design and implementation of Domain-Specific Compilers for Linear Algebra matrix equations.

Why?

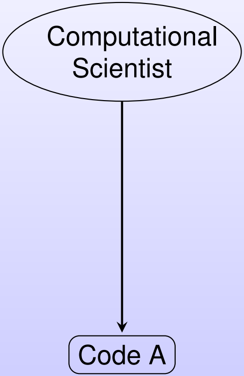
- Matrix equations are ubiquitous
- Complex and time consuming development
- It requires expertise from multiple areas:
  - Application domain
  - Numerics, algorithmics
  - High-performance computing

We are facing a productivity problem

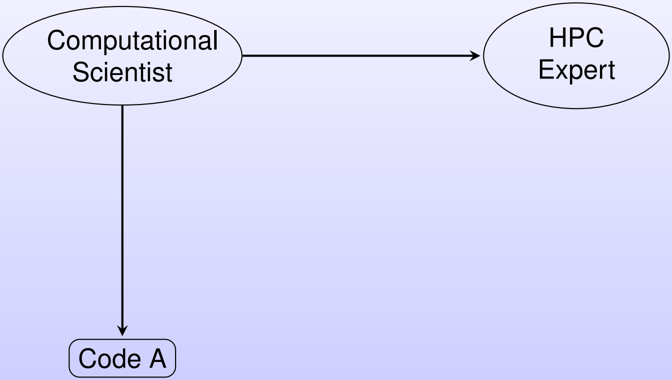
The productivity problem illustrated



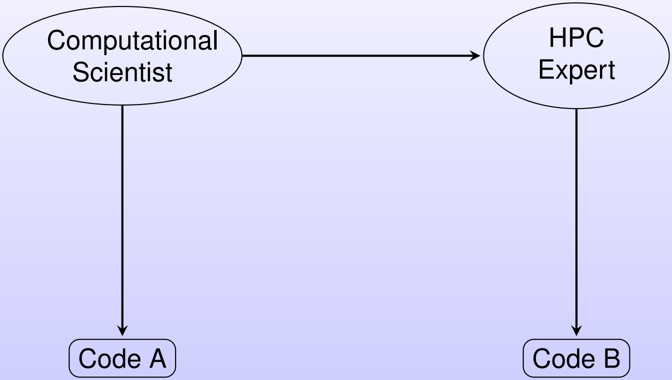
The productivity problem illustrated



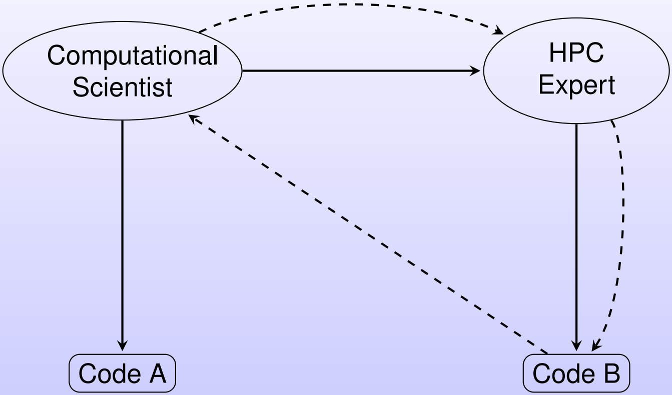
The productivity problem illustrated



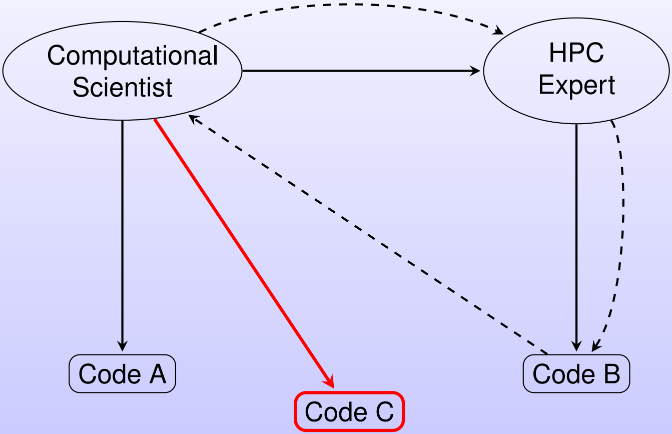
The productivity problem illustrated



The productivity problem illustrated



The productivity problem illustrated



Genome-Wide Association Study

$$b_i := (X_i^T M^{-1} X_i)^{-1} X_i^T M^{-1} y \quad 1 \leq i \leq m$$

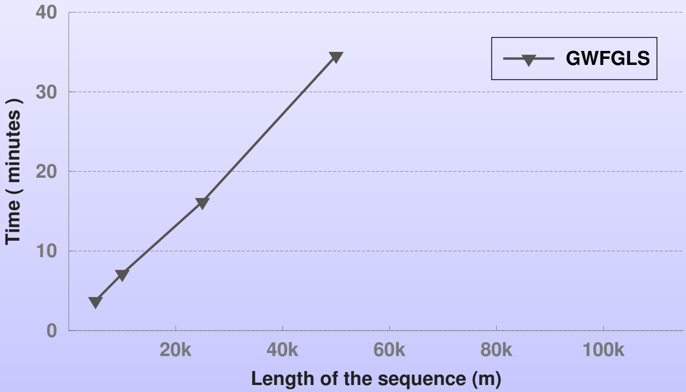
$$M \in R^{n \times n}, X \in R^{n \times p}, y \in R^n, b \in R^p$$

Example 1

Genome-Wide Association Study

$$b_i := (X_i^T M^{-1} X_i)^{-1} X_i^T M^{-1} y \quad 1 \le i \le m$$

$$M \in R^{n \times n}, X \in R^{n \times p}, y \in R^n, b \in R^p$$

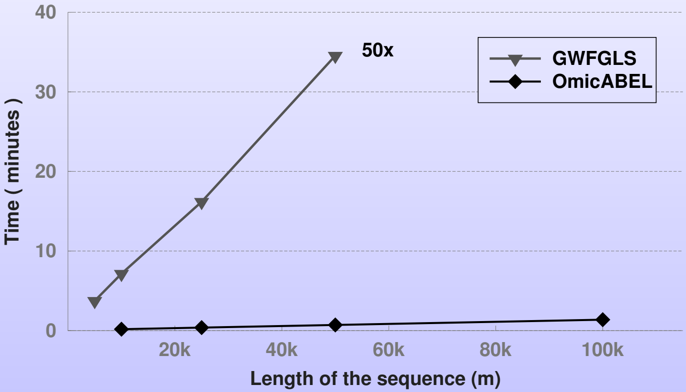


Example 1

Genome-Wide Association Study

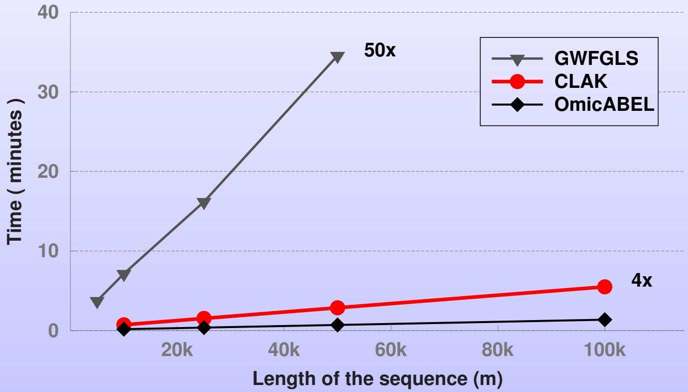
$$b_i := (X_i^T M^{-1} X_i)^{-1} X_i^T M^{-1} y \quad 1 \le i \le m$$

$$M \in R^{n \times n}, X \in R^{n \times p}, y \in R^n, b \in R^p$$



$$b_i := (X_i^T M^{-1} X_i)^{-1} X_i^T M^{-1} y \quad 1 \le i \le m$$

$$M \in R^{n \times n}, X \in R^{n \times p}, y \in R^n, b \in R^p$$



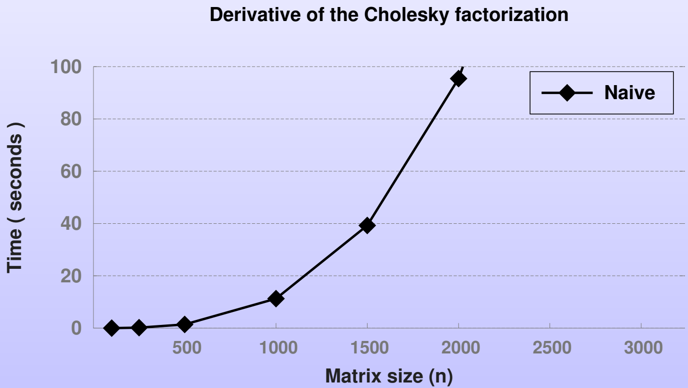
- $f : LL^T = A \quad \rightarrow \quad$ (LAPACK, FLAME, ...)

Example 2

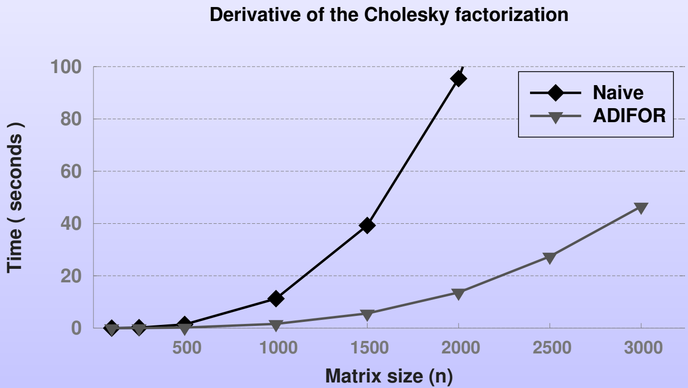
Derivative of the Cholesky factorization

- $f : LL^T = A \quad \rightarrow \quad$ (LAPACK, FLAME, ...)
- $f' : L'L^T + LL'^T = A' \quad \rightarrow \quad$?

- $f : LL^T = A \quad \rightarrow \quad$ (LAPACK, FLAME, ...)
- $f' : L'L^T + LL'^T = A' \quad \rightarrow \quad$?

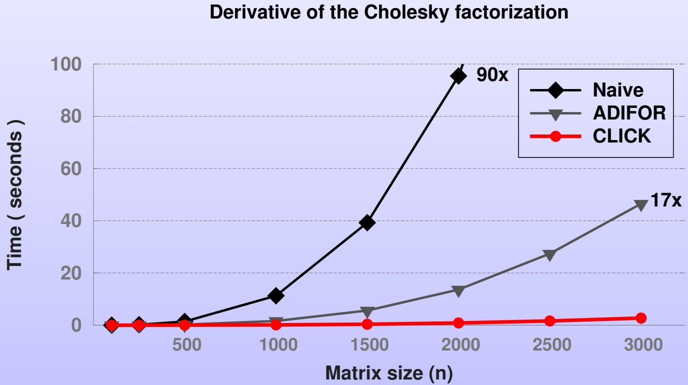


- $f : LL^T = A \quad \rightarrow \quad$ (LAPACK, FLAME, ...)
- $f' : L'L^T + LL'^T = A' \quad \rightarrow \quad$?



Derivative of the Cholesky factorization

- $f : LL^T = A \quad \rightarrow \quad$ (LAPACK, FLAME, ...)
- $f' : L'L^T + LL'^T = A' \quad \rightarrow \quad$?



The Goal

- Allow scientists to reason at the matrix equation level
- Relieve them from designing algorithms and writing code

The Goal

- Allow scientists to reason at the matrix equation level
- Relieve them from designing algorithms and writing code

- High-level language/interface: Equation + Knowledge

The Goal

- Allow scientists to reason at the matrix equation level
- Relieve them from designing algorithms and writing code

- High-level language/interface: Equation + Knowledge
- Our compilers take care of:
  - Deriving efficient algorithms that exploit the available knowledge
  - Generating code that takes advantage of kernels from high-performance libraries

The Goal

- Allow scientists to reason at the matrix equation level
- Relieve them from designing algorithms and writing code

- High-level language/interface: Equation + Knowledge
- Our compilers take care of:
  - Deriving efficient algorithms that exploit the available knowledge
  - Generating code that takes advantage of kernels from high-performance libraries

Productivity (+ Performance)

- Target: High-level equations

1st Compiler: CLAK

- Target: High-level equations
- Main idea: Decomposition onto building blocks

$X := S^{-1} \quad \longrightarrow$

Algorithm 2 : $X := S^{-1}$.

1: $LL^T = S$ (Cholesky factorization)
2: $L := L^{-1}$ (Triangular inverse)
3: $X := L^T L$ (Matrix product)

- Target: High-level equations
- Main idea: Decomposition onto building blocks
- Methodology: Replicate the reasoning of a human expert

$X := S^{-1} \quad \longrightarrow$

Algorithm 3 : $X := S^{-1}$.

1: $LL^T = S$ (Cholesky factorization)
2: $L := L^{-1}$ (Triangular inverse)
3: $X := L^T L$ (Matrix product)

2nd Compiler: CL1CK

- Target: Building blocks

2nd Compiler: CL1CK

- Target: Building blocks
- Core idea: Loop-based blocked algorithms

$$LL^T = A \quad \longrightarrow$$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0

While $n(A_{TL}) < n(A)$ **do**

$$\left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & \star & \star \\ \hline A_{10} & A_{11} & \star \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

$$\begin{aligned} A_{11} &= \Gamma(A_{11}) \\ A_{21} &= A_{21}\,\mathsf{TRIL}(A_{11})^{-T} \\ A_{22} &= A_{22} - A_{21}A_{21}^T \end{aligned}$$

$$\left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & \star & \star \\ \hline A_{10} & A_{11} & \star \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

endwhile

- Target: Building blocks
- Core idea: Loop-based blocked algorithms
- Methodology: FLAME Project's methodology

$$LL^T = A \quad \longrightarrow$$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0

While $n(A_{TL}) < n(A)$ **do**

$$\left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & \star & \star \\ \hline A_{10} & A_{11} & \star \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

$$\begin{aligned} A_{11} &= \Gamma(A_{11}) \\ A_{21} &= A_{21}\mathsf{TRIL}(A_{11})^{-T} \\ A_{22} &= A_{22} - A_{21}A_{21}^T \end{aligned}$$

$$\left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & \star & \star \\ \hline A_{10} & A_{11} & \star \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

endwhile

CLAK: The Input

- Operand declaration
  - Type: Matrix, Vector, Scalar
  - Properties: LowerTriangular, UpperTriangular, Symmetric, FullRank, ...

- Operand declaration
  - Type: Matrix, Vector, Scalar
  - Properties: LowerTriangular, UpperTriangular, Symmetric, FullRank, ...
- Operation
  - Operators: +, -, *, -1, T

CLAK: The Input

- Operand declaration
  - Type: Matrix, Vector, Scalar
  - Properties: LowerTriangular, UpperTriangular, Symmetric, FullRank, ...
- Operation
  - Operators: +, -, *, -1, T
  - `<lhs> = <expression>`
  - Any valid combination of operands and operators

- Operand declaration
  - Type: Matrix, Vector, Scalar
  - Properties: LowerTriangular, UpperTriangular, Symmetric, FullRank, ...
- Operation
  - Operators: +, -, *, -1, T
  - `<lhs> = <expression>`
  - Any valid combination of operands and operators
  - Operands may be labeled with subscripts (sequences of problems)

CLAK: The Input

- Operand declaration
  - Type: Matrix, Vector, Scalar
  - Properties: LowerTriangular, UpperTriangular, Symmetric, FullRank, ...
- Operation
  - Operators: +, -, *, -1, T
  - `<lhs> = <expression>`
  - Any valid combination of operands and operators
  - Operands may be labeled with subscripts (sequences of problems)

```
Equation SEQ_OLS
  # Operands declaration
  Vector b   <Output>;
  Matrix X   <Input, FullRank, ColumnPanel>;
  Vector y   <Input>;
  # Equation
  b{i} = inv( trans(X{i}) * X{i} ) * trans(X{i}) * y
```

The same way that a traditional compiler...

- ... breaks a program into **assembly instructions** ...
- ... directly supported by **the processor** ...
- ... attempting different types of **optimizations**,

The same way that a traditional compiler...

- ... breaks a program into **assembly instructions** ...
- ... directly supported by **the processor** ...
- ... attempting different types of **optimizations**,

CLAK ...

- ... breaks a target operation down to **building blocks** ...
- ... directly supported by **high-performance libraries**, ...
- ... **tailoring** the algorithm **to the application**.

Traditional Comp.

- `ADD`
- `MUL`
- `DIV`
- `XOR`

Linear Algebra Comp.

- $LU = A$ (`LU`)
- $LL^T = A$ (`Cholesky`)
- $ZWZ^T = A$ (`Eigendec`)

- $C := AB$ (`MM`)
- $y := Ax$ (`MV`)
- $Ax = b$ (`TRSV`)

CLAK aims at replicating the reasoning of a human expert

CLAK aims at replicating the reasoning of a human expert

We ...

- studied the steps an expert takes,
- encoded them in a set of heuristics, and
- incorporated these heuristics into CLAK

The inverse operator receives a special treatment

The inverse operator receives a special treatment

$$x := A^{-1}b$$

The inverse operator receives a special treatment

$$x := A^{-1} b$$

1: $C := A^{-1}$
2: $x := C\ b$

The inverse operator receives a special treatment

$$x := A^{-1} b$$

1: $\cancel{C := A^{-1}}$
2: $\cancel{x := C\, b}$

1: $LU = A$ (GETRF)

The inverse operator receives a special treatment

$$x := (LU)^{-1} b$$

~~1: $C := A^{-1}$~~
~~2: $x := C\,b$~~

1: $LU = A$ (GETRF)

CLAK: Heuristics

The inverse operator receives a special treatment

$$x := U^{-1}L^{-1}b$$

~~1: $C := A^{-1}$~~
~~2: $x := C\,b$~~

1: $LU = A$ (GETRF)

The inverse operator receives a special treatment

$$x := U^{-1}L^{-1}b$$

~~1: $C := A^{-1}$~~
~~2: $x := C\,b$~~

1: $LU = A$ (GETRF)
2: $y := L^{-1}b$ (TRSV)

The inverse operator receives a special treatment

$$x := U^{-1}L^{-1}b$$

1: $\cancel{C := A^{-1}}$
2: $\cancel{x := C\,b}$

1: $LU = A$ (GETRF)
2: $y := L^{-1}b$ (TRSV)
3: $x := U^{-1}y$ (TRSV)

The inverse operator receives a special treatment

$$x := U^{-1}L^{-1}b$$

~~1: $C := A^{-1}$~~
~~2: $x := C\,b$~~

1: $LU = A$ (GETRF)
2: $y := L^{-1}b$ (TRSV)
3: $x := U^{-1}y$ (TRSV)

- Do not invert unless really required

CLAK: Heuristics

Identify opportunities for optimizations (i.e., reducing the complexity)

Identify opportunities for optimizations (i.e., reducing the complexity)

$$\alpha := y^T L^{-1} L^{-T} y$$

Identify opportunities for optimizations (i.e., reducing the complexity)

$$\alpha := y^T L^{-1} L^{-T} y$$

Computation reuse

1: $x := L^{-T} y$
2: $\alpha := x^T x$

Identify opportunities for optimizations (i.e., reducing the complexity)

$$y := ABx$$

Identify opportunities for optimizations (i.e., reducing the complexity)

$$y := ABx$$

Algorithm 1

1: $C := AB \quad O(n^3)$
2: $y := Cx \quad O(n^2)$

Identify opportunities for optimizations (i.e., reducing the complexity)

$$y := ABx$$

Algorithm 1

1: $C := AB \quad O(n^3)$
2: $y := Cx \quad O(n^2)$

Algorithm 2

1: $t := B\,x \quad O(n^2)$
2: $y := A\,t \quad O(n^2)$

Reduced dimensionality

Identify opportunities for optimizations (i.e., reducing the complexity)

$$y := ABx$$

Algorithm 1

1: $C := AB \quad O(n^3)$
2: $y := Cx \quad O(n^2)$

Algorithm 2

1: $t := B\,x \quad O(n^2)$
2: $y := A\,t \quad O(n^2)$

Priorities: Matrix-Vector over Matrix-Matrix

CLAK: Tree of decompositions

CLAK applies these heuristics mechanically:

CLAK: Tree of decompositions

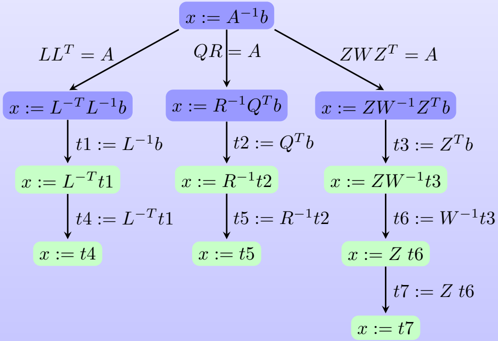
CLAK applies these heuristics mechanically:

- Process inverses until only applied to matrices in factored form
- Then, map the resulting expressions onto kernels

CLAK applies these heuristics mechanically:

- Process inverses until only applied to matrices in factored form
- Then, map the resulting expressions onto kernels

As a result, CLAK generates a tree of decompositions:



Factorizations

Matrix Property	**Factorizations**
Symmetric	LDL, QR, Eigendecomposition
SPD	Cholesky, QR, Eigendecomposition
Column Panel	QR
...	...

Reducing complexity

- Prioritization based on dimensionality of the operands:

#	Kernels	Example
1	inner product	$\alpha := x^T y$
2	matrix-vector operations	$y := Ax$, $b := L^{-1}x$
3	matrix-matrix operations	$C := AB$, $B := L^{-1}A$
4	outer product	$A := xy^T$
5	inversion of a triangular matrix	$C := L^{-1}$

- Common segments ($\alpha := y^T L^{-1} L^{-T} y$)

Built in a modular fashion

- Matrix algebra
- Inference of knowledge
- Other modules

$$\begin{aligned}
(A \times B)^T &\rightarrow B^T \times A^T \\
(A \times B)^{-1} \wedge \text{Square}(A) \wedge \text{Square}(B) &\rightarrow B^{-1} \times A^{-1} \\
Q^T \times Q \wedge \text{Orthogonal}(Q) &\rightarrow I \\
A^{-1} \times A &\rightarrow I \\
A \times I \wedge \text{Matrix}(A) &\rightarrow A
\end{aligned}$$

$$
\begin{array}{rcl}
(A \times B)^T & \rightarrow & B^T \times A^T \\
(A \times B)^{-1} \wedge \text{Square}(A) \wedge \text{Square}(B) & \rightarrow & B^{-1} \times A^{-1} \\
Q^T \times Q \wedge \text{Orthogonal}(Q) & \rightarrow & I \\
A^{-1} \times A & \rightarrow & I \\
A \times I \wedge \text{Matrix}(A) & \rightarrow & A
\end{array}
$$

$$
((QR)^T QR)^{-1} (QR)^T L^{-1} y \quad \longrightarrow \quad R^{-1} Q^T L^{-1} y
$$

$$
\begin{aligned}
(A \times B)^T &\rightarrow B^T \times A^T \\
(A \times B)^{-1} \wedge \text{Square}(A) \wedge \text{Square}(B) &\rightarrow B^{-1} \times A^{-1} \\
Q^T \times Q \wedge \text{Orthogonal}(Q) &\rightarrow I \\
A^{-1} \times A &\rightarrow I \\
A \times I \wedge \text{Matrix}(A) &\rightarrow A
\end{aligned}
$$

$$
\begin{aligned}
((QR)^T QR)^{-1}(QR)^T L^{-1} y &\longrightarrow R^{-1} Q^T L^{-1} y \\
(ZWZ^T + I)^{-1} &\longrightarrow Z(W + I)^{-1} Z^T
\end{aligned}
$$

$$
\begin{array}{rcl}
(A \times B)^T & \rightarrow & B^T \times A^T \\
(A \times B)^{-1} \wedge \text{Square}(A) \wedge \text{Square}(B) & \rightarrow & B^{-1} \times A^{-1} \\
Q^T \times Q \wedge \text{Orthogonal}(Q) & \rightarrow & I \\
A^{-1} \times A & \rightarrow & I \\
A \times I \wedge \text{Matrix}(A) & \rightarrow & A
\end{array}
$$

$$
\begin{array}{rcl}
((QR)^T QR)^{-1}(QR)^T L^{-1} y & \longrightarrow & R^{-1} Q^T L^{-1} y \\
(ZWZ^T + I)^{-1} & \longrightarrow & Z(W + I)^{-1} Z^T
\end{array}
$$

About 50 such rules

Type of operand:

Expression	Constraint	Inferred Property
$A \times B$	`Matrix(A)` $\wedge$ `Matrix(B)`	Matrix($A \times B$)
$A \times x$	`Matrix(A)` $\wedge$ `Vector(x)`	Vector($A \times x$)
$x^T \times y$	`Vector(x)` $\wedge$ `Vector(y)`	Scalar($x^T \times y$)

Type of operand:

Expression	**Constraint**	**Inferred Property**
$A \times B$	`Matrix(A)` $\wedge$ `Matrix(B)`	Matrix($A \times B$)
$A \times x$	`Matrix(A)` $\wedge$ `Vector(x)`	Vector($A \times x$)
$x^T \times y$	`Vector(x)` $\wedge$ `Vector(y)`	Scalar($x^T \times y$)

Matrix factorizations (properties of factors):

QR ($QR = A$):

Input	A: matrix, column-panel, full rank
Output	Q: matrix, orthogonal, column-panel, full rank
	R: matrix, square, upper triangular, full rank

Building blocks:

Expression	Constraint	Inferred Property
$S_1 + \cdots + S_n$	$\forall_i$ Symmetric(S_i)	Symmetric
$-S$	Symmetric(S)	Symmetric
S^T	Symmetric(S)	Symmetric
S^{-1}	Symmetric(S)	Symmetric
expr	expr $==$ expr^T	Symmetric

Building blocks:

Expression	**Constraint**	**Inferred Property**
$S_1 + \cdots + S_n$	$\forall_i$ Symmetric(S_i)	Symmetric
$-S$	Symmetric(S)	Symmetric
S^T	Symmetric(S)	Symmetric
S^{-1}	Symmetric(S)	Symmetric
expr	expr $==$ expr^T	Symmetric

More than a hundred such rules!

- Sequences of problems

Algorithm 4 CLAK-EIG

$Z \Lambda Z^T = \Phi$
for $i := 1$ **to** m **do**
    $K_i := X_i^T Z$
end for
for $j := 1$ **to** t **do**
    $D_j := h_j \Lambda + (1 - h_j) I$
    $y_j := Z^T y_j$
    for $i := 1$ **to** m **do**
        $V_{ij} := K_i D_j^{-1}$
        $A_{ij} := V_{ij} K_i^T$
        $Q_{ij} R_{ij} = A_{ij}$
        $b_{ij} := V_{ij} y_j$
        $b_{ij} := Q_{ij}^T b_{ij}$
        $b_{ij} := R_{ij}^{-1} b_{ij}$
    end for
end for

- Derivative Operator

`z = alpha * x + y` $\longrightarrow$ `dv(z = alpha * x + y)` ?:

- `dv(z) = dv(alpha) * x + alpha * dv(x) + dv(y)`
- `dv(z) = dv(alpha) * x + alpha * dv(x)`
- `dv(z) = dv(alpha) * x`
- ...

- Matlab Code

```
function [b] = GWAS_26_2(X, y, h, Phi, sm, sn, nXs, nys)
   b = zeros(sm, nXs * nys);
   T3 = zeros(sm, sn * nXs);
   [Z1, W1] = eig( Phi );

   for i = 1:nXs
      T3(:, sn*(i-1)+1:sn*i) = X(:, sm*(i-1)+1:sm*i)' * Z1;
   end
   for j = 1:nys
      T1 = 1 * eye(sn) + - h(j) * eye(sn);
      T2 =  T1 + h(j) * W1;
      T6 =  Z1' * y(:, j);
      for i = 1:nXs
          [...]
```

Additional modules

- Fortran Code

```
SUBROUTINE GWAS_26_2( X, csX, dsX, y, csy, h, Phi, csPhi,
                      b, csb, sn, sm, nXs, nys )
   INTEGER sn, sm, nXs, nys, csX, dsX, csy, csPhi, csb
   [...]
   call dsyevr( 'V', 'A', 'L', sn, Phi( 1, 1 ), sn, ddummy,
                ddummy, idummy, idummy, ddummy, nCompPairs5,
                W8( 1 ), Z7( 1, 1 ), sn, isuppz4, ... )
   DO i = 1, nXs
      call dgemm( 'T', 'N', sm, sn, sn, ONE, X( 1, 1, i ),
                  sn, Z7( 1, 1 ), sn, ZERO, tmp70( 1, 1, i ), sm )
   END DO
   DO j = 1, nys
      DO iter1 = 1, sn
         tmp28( iter1 ) = 1 + ( - h(j))
         [...]
```

Sample of targeted operations

- Matrix inversions
- Multiple linear systems
- Ordinary least-squares
- Generalized least-squares
- Sequences of problems

Sample of targeted operations

- Matrix inversions
- Multiple linear systems
- Ordinary least-squares
- Generalized least-squares
- Sequences of problems

- Equations arising in Genome-wide association studies
- Derivatives of matrix products
- Derivatives of linear systems

FLAME Methodology

- More than a decade of development
- Many algorithms derived by hand and incorporated into libraries (FLAME, Elemental)
- Manual derivation becomes tedious and error prone when the complexity of the equations increases

CL1CK demonstrates...

- FLAME methodology can be applied automatically
- The broad applicability of the methodology

Operations are described by means of two predicates:

The *Precondition* (P_{pre}) and the *Postcondition* (P_{post}).

Operations are described by means of two predicates:

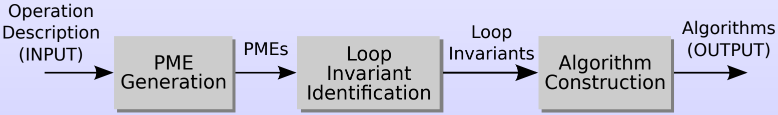
The *Precondition* (P_{pre}) and the *Postcondition* (P_{post}).

Example: Derivative of Cholesky

$$G = gChol(L, B) \equiv \begin{cases} P_{\text{pre}} : \{ \texttt{Output}(G) \wedge \texttt{Input}(L) \wedge \texttt{Input}(B) \wedge \\ \qquad \texttt{Matrix}(G) \wedge \texttt{Matrix}(L) \wedge \texttt{Matrix}(B) \wedge \\ \qquad \texttt{LowerTriangular}(G) \wedge \texttt{Symmetric}(B) \wedge \\ \qquad \texttt{LowerTriangular}(L) \} \\ \\ P_{\text{post}} : \{ GL^T + LG^T = B \} \end{cases}$$

CL1CK: The Methodology

- CL1CK implements the FLAME methodology in 3 stages:



CL1CK: Input

$$G = gChol(L, B) \equiv \begin{cases} P_{\text{pre}} : \{\texttt{Output}(G) \wedge \texttt{Input}(L) \wedge \texttt{Input}(B) \wedge \\ \qquad \texttt{Matrix}(G) \wedge \texttt{Matrix}(L) \wedge \texttt{Matrix}(B) \wedge \\ \qquad \texttt{LowerTriangular}(G) \wedge \texttt{Symmetric}(B) \wedge \\ \qquad \texttt{LowerTriangular}(L)\} \\ \\ P_{\text{post}} : \{GL^T + LG^T = B\} \end{cases}$$

$$G = gChol(L, B) \equiv \begin{cases} P_{\text{pre}} : \{\texttt{Output}(G) \ \wedge \ \texttt{Input}(L) \ \wedge \texttt{Input}(B) \ \wedge \\ \qquad\quad \texttt{Matrix}(G) \ \wedge \ \texttt{Matrix}(L) \ \wedge \texttt{Matrix}(B) \ \wedge \\ \qquad\quad \texttt{LowerTriangular}(G) \ \wedge \ \texttt{Symmetric}(B) \ \wedge \\ \qquad\quad \texttt{LowerTriangular}(L)\} \\ \\ P_{\text{post}} : \{GL^T + LG^T = B\} \end{cases}$$

$$\Downarrow$$

```
equal[
    plus[ times[ G_, trans[L_] ], times[ L_, trans[G_] ] ], B_
] /; isInputQ[L] && isInputQ[B] && isOutputQ[G] &&
     isMatrixQ[L] && isMatrixQ[B] && isMatrixQ[G] &&
     isLowerTriQ[L] && isSymmetricQ[B] && isLowerTriQ[G]
```

Partitioned Matrix Expressions are generated in 3 steps:

1. Decompose the problem into smaller ones
2. Find out how to solve the sub-problems
3. Combine the solutions

$$GL^T + LG^T = B$$

CL1CK: PME Generation

Decomposition into subproblems

$$GL^T + LG^T = B$$

$$\Downarrow$$

$$\left(\begin{array}{c|c} G_{TL} & 0 \\ \hline G_{BL} & G_{BR} \end{array}\right)\left(\begin{array}{c|c} L_{TL}^T & L_{BL}^T \\ \hline 0 & L_{BR}^T \end{array}\right)+\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)\left(\begin{array}{c|c} G_{TL}^T & G_{BL}^T \\ \hline 0 & G_{BR}^T \end{array}\right)=\left(\begin{array}{c|c} B_{TL} & B_{BL}^T \\ \hline B_{BL} & B_{BR} \end{array}\right)$$

$$GL^T + LG^T = B$$

$$\Downarrow$$

$$\left(\begin{array}{c|c} G_{TL} & 0 \\ \hline G_{BL} & G_{BR} \end{array}\right)\left(\begin{array}{c|c} L_{TL}^T & L_{BL}^T \\ \hline 0 & L_{BR}^T \end{array}\right) + \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)\left(\begin{array}{c|c} G_{TL}^T & G_{BL}^T \\ \hline 0 & G_{BR}^T \end{array}\right) = \left(\begin{array}{c|c} B_{TL} & B_{BL}^T \\ \hline B_{BL} & B_{BR} \end{array}\right)$$

$$\Downarrow$$

$$\left(\begin{array}{c|c} G_{TL}L_{TL}^T + L_{TL}G_{TL}^T = B_{TL} & * \\ \hline G_{BL}L_{TL}^T + L_{BL}G_{TL}^T = B_{BL} & G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T = B_{BR} \end{array}\right)$$

Cl1ck: PME Generation

Identifying the subproblems

$$\left(\begin{array}{c|c} G_{TL}L_{TL}^T + L_{TL}G_{TL}^T = B_{TL} & * \\ \hline G_{BL}L_{TL}^T + L_{BL}G_{TL}^T = B_{BL} & G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T = B_{BR} \end{array}\right)$$

$$\left(\begin{array}{c|c} G_{TL}L_{TL}^T + L_{TL}G_{TL}^T = B_{TL} & * \\ \hline G_{BL}L_{TL}^T + L_{BL}G_{TL}^T = B_{BL} & G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T = B_{BR} \end{array}\right)$$

$$\Downarrow$$

$$\left(\begin{array}{c|c} G_{TL} := gChol(L_{TL}, B_{TL}) & * \\ \hline G_{BL}L_{TL}^T + L_{BL}G_{TL}^T = B_{BL} & G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T = B_{BR} \end{array}\right)$$

Cl1ck: PME Generation

Identifying the subproblems

$$\left(\begin{array}{c|c} G_{TL} := gChol(L_{TL}, B_{TL}) & * \\ \hline G_{BL}L_{TL}^T + L_{BL}G_{TL}^T = B_{BL} & G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T = B_{BR} \end{array}\right)$$

$$\left(\begin{array}{c|c} G_{TL} := gChol(L_{TL}, B_{TL}) & * \\ \hline G_{BL}L_{TL}^T + L_{BL}G_{TL}^T = B_{BL} & G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T = B_{BR} \end{array}\right)$$

$$\Downarrow$$

$$\left(\begin{array}{c|c} G_{TL} := gChol(L_{TL}, B_{TL}) & * \\ \hline G_{BL}L_{TL}^T = B_{BL} - L_{BL}G_{TL}^T & G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T = B_{BR} \end{array}\right)$$

Identifying the subproblems

$$\left(\begin{array}{c|c} G_{TL} := gChol(L_{TL}, B_{TL}) & * \\ \hline G_{BL}L_{TL}^T = B_{BL} - L_{BL}G_{TL}^T & G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T = B_{BR} \end{array}\right)$$

Identifying the subproblems

$$\left(\begin{array}{c|c} G_{TL} := gChol(L_{TL}, B_{TL}) & * \\ \hline G_{BL}L_{TL}^T = B_{BL} - L_{BL}G_{TL}^T & G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T = B_{BR} \end{array}\right)$$

$$\Downarrow$$

$$\left(\begin{array}{c|c} G_{TL} := gChol(L_{TL}, B_{TL}) & * \\ \hline G_{BL} := (B_{BL} - L_{BL}G_{TL}^T)L_{TL}^{-T} & G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T = B_{BR} \end{array}\right)$$

$$\left(\begin{array}{c|c} G_{TL} := gChol(L_{TL}, B_{TL}) & * \\ \hline G_{BL} := (B_{BL} - L_{BL}G_{TL}^T)L_{TL}^{-T} & G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T = B_{BR} \end{array}\right)$$

$$\left(\begin{array}{c|c} G_{TL} := gChol(L_{TL}, B_{TL}) & * \\ \hline G_{BL} := (B_{BL} - L_{BL}G_{TL}^T)L_{TL}^{-T} & G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T = B_{BR} \end{array}\right)$$

$$\Downarrow$$

$$\left(\begin{array}{c|c} G_{TL} := gChol(L_{TL}, B_{TL}) & * \\ \hline G_{BL} := (B_{BL} - L_{BL}G_{TL}^T)L_{TL}^{-T} & G_{BR}L_{BR}^T + L_{BR}G_{BR}^T = B_{BR} - G_{BL}L_{BL}^T - L_{BL}G_{BL}^T \end{array}\right)$$

$$\left(\begin{array}{c|c} G_{TL} := gChol(L_{TL}, B_{TL}) & * \\ \hline G_{BL} := (B_{BL} - L_{BL} G_{TL}^T) L_{TL}^{-T} & G_{BR} L_{BR}^T + L_{BR} G_{BR}^T = B_{BR} - G_{BL} L_{BL}^T - L_{BL} G_{BL}^T \end{array}\right)$$

Cl1ck: PME Generation

The PME

$$\left(\begin{array}{c|c} G_{TL} := gChol(L_{TL}, B_{TL}) & * \\ \hline G_{BL} := (B_{BL} - L_{BL}G_{TL}^T)L_{TL}^{-T} & G_{BR}L_{BR}^T + L_{BR}G_{BR}^T = B_{BR} - G_{BL}L_{BL}^T - L_{BL}G_{BL}^T \end{array}\right)$$

$$\Downarrow$$

$$\boxed{\left(\begin{array}{c|c} G_{TL} := gChol(L_{TL}, B_{TL}) & * \\ \hline G_{BL} := (B_{BL} - L_{BL}G_{TL}^T)L_{TL}^{-T} & G_{BR} := gChol(L_{BR}, B_{BR} - G_{BL}L_{BL}^T - L_{BL}G_{BL}^T) \end{array}\right)}$$

Triangular Sylvester Equation ($AX + XB = C$):

$$\left(\begin{array}{c|c} X_{TL} := \Omega(A_{TL}, B_{TL}, C_{TL}) & X_{TR} := \Omega(A_{TL}, B_{BR}, C_{TR} - X_{TL}B_{TR}) \\ \hline X_{BL} := \Omega(A_{BR}, B_{TL}, C_{BL} - A_{BL}X_{TL}) & \begin{array}{l} X_{BR} := \Omega(A_{BR}, B_{BR}, \\ \qquad\qquad C_{BR} - X_{BL}B_{TR} - A_{BL}X_{TR}) \end{array} \end{array}\right)$$

Triangular Sylvester Equation ($AX + XB = C$):

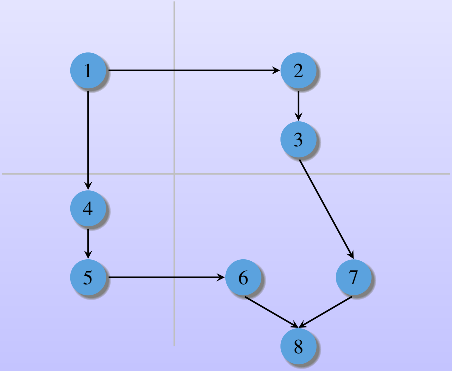
$$\left(\begin{array}{c|c} X_{TL} := \Omega(A_{TL}, B_{TL}, C_{TL}) & X_{TR} := \Omega(A_{TL}, B_{BR}, C_{TR} - X_{TL}B_{TR}) \\ \hline X_{BL} := \Omega(A_{BR}, B_{TL}, C_{BL} - A_{BL}X_{TL}) & \begin{array}{l} X_{BR} := \Omega(A_{BR}, B_{BR}, \\ \qquad C_{BR} - X_{BL}B_{TR} - A_{BL}X_{TR}) \end{array} \end{array}\right)$$

Triangular Lyapunov Equation ($AX + XA^T = C$):

$$\left(\begin{array}{c|c} X_{TL} := \Lambda(A_{TL}, C_{TL}) & * \\ \hline X_{BL} := \Omega(A_{BR}, A_{TL}^T, C_{BL} - A_{BL}X_{TL}) & \begin{array}{l} X_{BR} := \Lambda(A_{BR}, \\ \qquad C_{BR} - X_{BL}A_{BL}^T - A_{BL}X_{BL}^T) \end{array} \end{array}\right)$$

Graph of dependencies

$$\left(\begin{array}{c|c} X_{TL} := \Omega(A_{TL}, B_{TL}, C_{TL}) & X_{TR} := \Omega(A_{TL}, B_{BR}, C_{TR} - X_{TL}B_{TR}) \\ \hline X_{BL} := \Omega(A_{BR}, B_{TL}, C_{BL} - A_{BL}X_{TL}) & \begin{array}{l} X_{BR} := \Omega(A_{BR}, B_{BR}, \\ \qquad C_{BR} - X_{BL}B_{TR} - A_{BL}X_{TR}) \end{array} \end{array}\right)$$



Partition $B \rightarrow \left(\begin{array}{c|c} B_{TL} & \star \\ \hline B_{BL} & B_{BR} \end{array}\right), L \rightarrow \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right), G \rightarrow \left(\begin{array}{c|c} G_{TL} & 0 \\ \hline G_{BL} & G_{BR} \end{array}\right)$

where B_{TL}, L_{TL}, and G_{TL} are 0×0

while $\texttt{size}(B_{TL}) < \texttt{size}(B)$ **do**

$$\left(\begin{array}{c|c} B_{TL} & \star \\ \hline B_{BL} & B_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} B_{00} & \star & \star \\ \hline B_{10} & B_{11} & \star \\ \hline B_{20} & B_{21} & B_{22} \end{array}\right), \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline L_{10} & L_{11} & 0 \\ \hline L_{20} & L_{21} & L_{22} \end{array}\right), \ldots$$

Variant 1

$G_{10} := B_{10} - L_{10} G_{00}^T$ (TRMM)
$G_{10} := G_{10} L_{00}^{-T}$ (TRSM)
$G_{11} := B_{11} - G_{10} L_{10}^T - L_{10} G_{10}^T$ (SYR2K)
$G_{11} :=$ gChol(G_{11}, L_{11}) (gCHOL)

Variant 2

$G_{10} := G_{10} L_{00}^{-T}$ (TRSM)
$G_{11} := B_{11} - G_{10} L_{10}^T - L_{10} G_{10}^T$ (SYR2K)
$G_{11} :=$ gChol(G_{11}, L_{11}) (gCHOL)
$G_{21} := B_{21} - L_{21} G_{11}^T$ (TRMM)
$G_{21} := G_{21} - L_{20} G_{10}^T$ (GEMM)

Variant 3

...

Variant 4

...

$$\left(\begin{array}{c|c} B_{TL} & \star \\ \hline B_{BL} & B_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} B_{00} & \star & \star \\ \hline B_{10} & B_{11} & \star \\ \hline B_{20} & B_{21} & B_{22} \end{array}\right), \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} L_{00} & 0 & 0 \\ \hline L_{10} & L_{11} & 0 \\ \hline L_{20} & L_{21} & L_{22} \end{array}\right), \ldots$$

endwhile

Routine 1: gCHOL. Variant 1.

```
void gChol_blk_var1( FLA_Obj G, FLA_Obj L, int nb )
{
  FLA_Obj GTL, GTR, GBL, GBR, G00, G01, G02, G10, G11, G12, G20, G21, G22;
  FLA_Obj LTL, LTR, LBL, LBR, L00, L01, L02, L10, L11, L12, L20, L21, L22;

  FLA_Part_2x2( G, &GTL, &GTR,
                   &GBL, &GBR, 0, 0, FLA_TL );
  [...]
  while ( FLA_Obj_length( GTL ) < FLA_Obj_length( G ) ) {
    FLA_Repart_2x2_to_3x3( GTL, GTR, &G00, &G01, &G02,
                                     &G10, &G11, &G12,
                           GBL, GBR, &G20, &G21, &G22, nb, nb, FLA_BR );
    [...]
    FLA_Trmmsx_external( FLA_RIGHT, FLA_LOWER_TRIANGULAR,
                         FLA_TRANSPOSE, FLA_NONUNIT_DIAG,
                         FLA_MINUS_ONE, G00, L10, FLA_ONE, G10);
    FLA_Trsm( FLA_RIGHT, FLA_LOWER_TRIANGULAR, FLA_TRANSPOSE,
              FLA_NONUNIT_DIAG, FLA_ONE, L00, G10);
    FLA_Syr2k( FLA_LOWER_TRIANGULAR, FLA_NO_TRANSPOSE,
               FLA_MINUS_ONE, G10, L10, FLA_ONE, G11 );
    FLA_gChol_unb(G11, L11);
  [...]
  }
}
```

Sample of targeted operations

- BLAS(-like)
  - BLAS 3: GEMM, SYMM, SYRK, TRSM, ...
  - BLAS 2: GEMV, SYMV, GER, TRSV, ...
  - BLAS 1: AXPY, DOT, ...
- LAPACK
  - Factorizations
  - Inverses
- RECSY
  - Continuous-time Sylvester
  - Continuous-time Lyapunov

Sample of targeted operations

- BLAS(-like)
  - BLAS 3: GEMM, SYMM, SYRK, TRSM, ...
  - BLAS 2: GEMV, SYMV, GER, TRSV, ...
  - BLAS 1: AXPY, DOT, ...
- LAPACK
  - Factorizations
  - Inverses
- RECSY
  - Continuous-time Sylvester
  - Continuous-time Lyapunov
- Derivatives of the above
  - Dv(Triangular solve)
  - Dv(Cholesky)
  - ...

1. Two Linear Algebra Compilers
2. CLAK
3. CL1CK
4. Contributions

Contributions

CLAK

- High-level matrix equations
- Decomposition onto building blocks
- Replicate reasoning of a human expert
- Search guided by knowledge
- Prototypes of code generators (Matlab, Fortran)

Contributions

CLAK

- High-level matrix equations
- Decomposition onto building blocks
- Replicate reasoning of a human expert
- Search guided by knowledge
- Prototypes of code generators (Matlab, Fortran)

CL1CK

- Building blocks
- Full automation of FLAME's methodology
- Dynamically increases its knowledge-base
- Potential to derive entire libraries of kernels

Contributions

Additional results

- Inference engine for dynamic deduction of knowledge/properties

Contributions

Additional results

- Inference engine for dynamic deduction of knowledge/properties
- Study: DSCs vs ADIFOR for differentiated BLAS and LAPACK ops

Contributions

Additional results

- Inference engine for dynamic deduction of knowledge/properties
- Study: DSCs vs ADIFOR for differentiated BLAS and LAPACK ops
- OmicABEL in GenABEL (large speedups, state-of-the-art)

Publications

- **CLAK**
  - Fabregat & Bientinesi. A domain-specific compiler for linear algebra operations. In *High-performance computing for computational science (VECPAR 2012)*.
  - Fabregat & Bientinesi. Application-tailored linear algebra algorithms. *International journal of high-performance computing and applications (IJHPCA)*, 2013.
- **CL1CK**
  - Fabregat & Bientinesi. Knowledge-based automatic generation of Partitioned Matrix Expressions. In *Computer Algebra in Scientific Computing (CASC 2011)*.
  - Fabregat & Bientinesi. Automatic generation of Loop Invariants for matrix operations. *International conference in computational science and its applications (CASA 2011)*.

Publications (II)

- **Computational Biology**
  - Fabregat, Aulchenko & Bientinesi. Solving sequencies of generalized least-squares problems on multi-threaded architectures. *Applied Mathematics and Computation journal (AMC)*. Accepted pending minor revision.
  - Fabregat & Bientinesi. Computing petaflops over terabytes of date: The case of genome-wide association studies. *ACM Transactions on Mathematical Software (TOMS)*.
- **Tensor Contractions**
  - Di Napoli, Fabregat, Quintana & Bientinesi. Towards an efficient use of the BLAS library for multilinear tensor contractions. *Applied Mathematics and Computation journal (AMC)*. Accepted pending minor revision.

Future research directions

- Integration of performance analysis techniques

Future research directions

- Integration of performance analysis techniques
- Algorithm analysis and code generation for parallel archs

- Integration of performance analysis techniques
- Algorithm analysis and code generation for parallel archs
- Extend the scope of CLAK

Future research directions

- Integration of performance analysis techniques
- Algorithm analysis and code generation for parallel archs
- Extend the scope of CLAK
- Further explore the potential of our DSCs on AD

Thanks to:

- Examination Committee
- HPAC
- Collaborators
- AICES Students and Service team
- RWTH Computing Center, DAAD

Financial support from the **Deutsche Forschungsgemeinschaft** (German Research Association) through grant GSC 111 is gratefully acknowledged.

Deutsche Forschungsgemeinschaft DFG