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
Introduction

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
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
Introduction

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
Introduction

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

Computational Scientist
The productivity problem illustrated

Computational Scientist

Code A
The productivity problem illustrated

Computational Scientist → Code A → HPC Expert
The productivity problem illustrated

Computational Scientist → Code A

HPC Expert → Code B
The productivity problem illustrated

Computational Scientist

HPC Expert

Code A

Code B

Diego Fabregat (AICES, RWTH Aachen)
The productivity problem illustrated

- Computational Scientist
- HPC Expert
- Code A
- Code B
- Code C
Example 1

Genome-Wide Association Study

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

\[ M \in \mathbb{R}^{n \times n}, \quad X \in \mathbb{R}^{n \times p}, \quad y \in \mathbb{R}^{n}, \quad b \in \mathbb{R}^{p} \]
Example 1

Genome-Wide Association Study

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

\[ M \in R^{n \times n}, X \in R^{n \times p}, y \in R^n, b \in R^p \]
Example 1
Genome-Wide Association Study

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

\[ M \in \mathbb{R}^{n \times n}, X \in \mathbb{R}^{n \times p}, y \in \mathbb{R}^n, b \in \mathbb{R}^p \]
Example 1

Genome-Wide Association Study

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

\[ M \in \mathbb{R}^{n \times n}, X \in \mathbb{R}^{n \times p}, y \in \mathbb{R}^n, b \in \mathbb{R}^p \]
Example 2

Derivative of the Cholesky factorization

\[ f : LL^T = A \rightarrow (\text{LAPACK, FLAME, ...}) \]
Example 2

Derivative of the Cholesky factorization

- \( f : LL^T = A \rightarrow \) (LAPACK, FLAME, ...
- \( f' : L'L^T + LL'T = A' \rightarrow ? \)
Example 2

Derivative of the Cholesky factorization

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

![Graph showing the derivative of the Cholesky factorization for matrix sizes and time taken.](image-url)
Example 2

Derivative of the Cholesky factorization

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

![Graph showing the derivative of the Cholesky factorization for different matrix sizes.](attachment:graph.png)
Example 2

Derivative of the Cholesky factorization

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

Diego Fabregat (AICES, RWTH Aachen)
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)
1. Two Linear Algebra Compilers

2. CLAK

3. CL1CK

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

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

\[ X := S^{-1} \]

Algorithm 3: \[ X := S^{-1}. \]

1. \[ LL^T = S \] (Cholesky factorization)
2. \[ L := L^{-1} \] (Triangular inverse)
3. \[ X := L^T L \] (Matrix product)
Target: Building blocks
2nd Compiler: CL1CK

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

\[ LL^T = A \]

Partition \( A \rightarrow \begin{pmatrix} A_{TL} & \star \\ A_{BL} & A_{BR} \end{pmatrix} \)

where \( A_{TL} \) is \( 0 \times 0 \)

While \( n(A_{TL}) < n(A) \) do

\[
\begin{pmatrix} A_{TL} & \star \\ A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & \star & \star \\ A_{10} & A_{11} & \star \\ A_{20} & A_{21} & A_{22} \end{pmatrix}
\]

\[
A_{11} = \Gamma(A_{11}) \\
A_{21} = A_{21} \text{TRIL}(A_{11})^{-T} \\
A_{22} = A_{22} - A_{21}A_{21}^T
\]

\[
\begin{pmatrix} A_{TL} & \star \\ A_{BL} & A_{BR} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & \star & \star \\ A_{10} & A_{11} & \star \\ A_{20} & A_{21} & A_{22} \end{pmatrix}
\]

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

\[
LL^T = A \quad \rightarrow
\]

**Partition**

\[
A \rightarrow \begin{pmatrix} 
A_{TL} & * \\
A_{BL} & A_{BR} 
\end{pmatrix}
\]

where \( A_{TL} \) is \( 0 \times 0 \)

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

\[
\begin{pmatrix} 
A_{TL} & * \\
A_{BL} & A_{BR} 
\end{pmatrix} \rightarrow \begin{pmatrix} 
A_{00} & * & * \\
A_{10} & A_{11} & * \\
A_{20} & A_{21} & A_{22} 
\end{pmatrix}
\]

\[
A_{11} = \Gamma(A_{11}) \\
A_{21} = A_{21} \text{TRIL}(A_{11})^{-T} \\
A_{22} = A_{22} - A_{21}A_{21}^T
\]

**endwhile**
Two Linear Algebra Compilers

CLAK

CL1CK

Contributions
Operand declaration
- Type: Matrix, Vector, Scalar
- Properties: LowerTriangular, UpperTriangular, Symmetric, FullRank, ...
CLAK: The Input

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

- Operation
  - Operators: +, -, *, -1, T

Equation SEQ_OLS

Vector b <Output>
Matrix X <Input, FullRank, ColumnPanel>
Vector y <Input>

b\{i\} = inv( trans(X\{i\}) * X\{i\} ) * trans(X\{i\}) * y
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

---

```c
Vector b <Output>
Matrix X <Input, FullRank, ColumnPanel>
Vector y <Input>

# Equation
b[i] = inv( trans(X[i]) * X[i] ) * trans(X[i]) * y
```

Diego Fabregat (AICES, RWTH Aachen)

Knowledge-Based Auto Gen of Algs and Code

December 6th, 2013 11 / 45
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)

---

Diego Fabregat (AICES, RWTH Aachen)

Knowledge-Based Auto Gen of Algs and Code

December 6th, 2013 11 / 45
**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**

```plaintext
# 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
```

Diego Fabregat (AICES, RWTH Aachen)
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.
Building blocks

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: The Approach

CLAK aims at replicating the reasoning of a human expert.
CLAK: The Approach

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: \( C := A^{-1} \)
2: \( x := C \cdot b \)

1: \( LU = A \) (GETRF)

Do not invert unless really required.
The inverse operator receives a special treatment

\[ x := (LU)^{-1} b \]

1. \( C := A^{-1} \)
2. \( x := C b \)
The inverse operator receives a special treatment

\[ x := U^{-1} L^{-1} b \]

1. \( C := A^{-1} \)  
2. \( x := C b \)
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)

Do not invert unless really required.
The inverse operator receives a special treatment

\[ x := U^{-1}L^{-1}b \]

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

\[ LU = A \quad \text{(GETRF)} \]
\[ y := L^{-1}b \quad \text{(TRSV)} \]
\[ x := U^{-1}y \quad \text{(TRSV)} \]
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
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: \( C := AB \) \( O(n^3) \)
2: \( y := Cx \) \( O(n^2) \)

**Algorithm 2**

1: \( t := Bx \) \( O(n^2) \)
2: \( y := At \) \( O(n^2) \)
### CLAK: Heuristics

**Reduced dimensionality**

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

\[ y := ABx \]

#### Algorithm 1
1. \( C := AB \) \( O(n^3) \)
2. \( y := Cx \) \( O(n^2) \)

#### Algorithm 2
1. \( t := Bx \) \( O(n^2) \)
2. \( y := At \) \( O(n^2) \)

**Priorities: Matrix-Vector over Matrix-Matrix**
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:

\[
\begin{align*}
&x := A^{-1}b \\
&LL^T = A \\
&x := L^{-T}L^{-1}b \\
&t1 := L^{-1}b \\
&t4 := L^{-T}t1 \\
&x := t4 \\
&t7 := Z t6 \\
&t := t7 \\
&QR = A \\
&x := R^{-1}QTb \\
&t2 := QTb \\
&t5 := R^{-1}t2 \\
&x := t5 \\
&t := t5 \\
&ZWZ^T = A \\
&x := ZW^{-1}Z^Tb \\
&t3 := Z^Tb \\
&t6 := W^{-1}t3 \\
&x := t6 \\
&t := t6 \\
&LL^T = A \\
&QR = A \\
&ZWZ^T = A
\end{align*}
\]
Phase 1: Dealing with the inverse operator

## Factorizations

<table>
<thead>
<tr>
<th>Matrix Property</th>
<th>Factorizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetric</td>
<td>LDL, QR, Eigendecomposition</td>
</tr>
<tr>
<td>SPD</td>
<td>Cholesky, QR, Eigendecomposition</td>
</tr>
<tr>
<td>Column Panel</td>
<td>QR</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Reducing complexity

- Prioritization based on dimensionality of the operands:

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

- Common segments ($\alpha := y^T L^{-1} L^{-T} y$)
Built in a modular fashion

- Matrix algebra
- Inference of knowledge
- Other modules
CLAK: The Compiler’s Engine

Modules: Matrix Algebra

\[(A \times B)^T \rightarrow B^T \times A^T\]

\[(A \times B)^{-1} \land \text{Square}(A) \land \text{Square}(B) \rightarrow B^{-1} \times A^{-1}\]

\[Q^T \times Q \land \text{Orthogonal}(Q) \rightarrow I\]

\[A^{-1} \times A \rightarrow I\]

\[A \times I \land \text{Matrix}(A) \rightarrow A\]
### CLAK: The Compiler’s Engine

#### Modules: Matrix Algebra

<table>
<thead>
<tr>
<th>Rule</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>((A \times B)^T) \rightarrow (B^T \times A^T)</td>
<td></td>
</tr>
<tr>
<td>((A \times B)^{-1} \land \text{Square}(A) \land \text{Square}(B)) \rightarrow (B^{-1} \times A^{-1})</td>
<td></td>
</tr>
<tr>
<td>(Q^T \times Q \land \text{Orthogonal}(Q)) \rightarrow (I)</td>
<td></td>
</tr>
<tr>
<td>(A^{-1} \times A) \rightarrow (I)</td>
<td></td>
</tr>
<tr>
<td>(A \times I \land \text{Matrix}(A)) \rightarrow (A)</td>
<td></td>
</tr>
</tbody>
</table>

\[ (((QR)^T QR)^{-1}(QR)^T L^{-1}y \rightarrow R^{-1}Q^T L^{-1}y \]
\[(A \times B)^T \rightarrow B^T \times A^T\]
\[(A \times B)^{-1} \land \text{Square}(A) \land \text{Square}(B) \rightarrow B^{-1} \times A^{-1}\]
\[Q^T \times Q \land \text{Orthogonal}(Q) \rightarrow I\]
\[A^{-1} \times A \rightarrow I\]
\[A \times I \land \text{Matrix}(A) \rightarrow A\]

\[((QR)^T QR)^{-1} (QR)^T L^{-1} y \rightarrow R^{-1} Q^T L^{-1} y\]
\[(ZW Z^T + I)^{-1} \rightarrow Z(W + I)^{-1} Z^T\]
CLAK: The Compiler’s Engine

Modules: Matrix Algebra

\[(A \times B)^T \rightarrow B^T \times A^T\]

\[(A \times B)^{-1} \land \text{Square}(A) \land \text{Square}(B) \rightarrow B^{-1} \times A^{-1}\]

\[Q^T \times Q \land \text{Orthogonal}(Q) \rightarrow I\]

\[A^{-1} \times A \rightarrow I\]

\[A \times I \land \text{Matrix}(A) \rightarrow A\]

\[((QR)^T QR)^{-1} (QR)^T L^{-1} y \rightarrow R^{-1} Q^T L^{-1} y\]

\[(ZWZ^T + I)^{-1} \rightarrow Z(W + I)^{-1} Z^T\]

About 50 such rules
## Type of operand:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Constraint</th>
<th>Inferred Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \times B$</td>
<td>$\text{Matrix}(A) \land \text{Matrix}(B)$</td>
<td>$\text{Matrix}(A \times B)$</td>
</tr>
<tr>
<td>$A \times x$</td>
<td>$\text{Matrix}(A) \land \text{Vector}(x)$</td>
<td>$\text{Vector}(A \times x)$</td>
</tr>
<tr>
<td>$x^T \times y$</td>
<td>$\text{Vector}(x) \land \text{Vector}(y)$</td>
<td>$\text{Scalar}(x^T \times y)$</td>
</tr>
</tbody>
</table>
Type of operand:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Constraint</th>
<th>Inferred Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \times B$</td>
<td>$\text{Matrix}(A) \land \text{Matrix}(B)$</td>
<td>$\text{Matrix}(A \times B)$</td>
</tr>
<tr>
<td>$A \times x$</td>
<td>$\text{Matrix}(A) \land \text{Vector}(x)$</td>
<td>$\text{Vector}(A \times x)$</td>
</tr>
<tr>
<td>$x^T \times y$</td>
<td>$\text{Vector}(x) \land \text{Vector}(y)$</td>
<td>$\text{Scalar}(x^T \times y)$</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Constraint</th>
<th>Inferred Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1 + \cdots + S_n$</td>
<td>$\forall_i \text{Symmetric}(S_i)$</td>
<td>Symmetric</td>
</tr>
<tr>
<td>$-S$</td>
<td>$\text{Symmetric}(S)$</td>
<td>Symmetric</td>
</tr>
<tr>
<td>$S^T$</td>
<td>$\text{Symmetric}(S)$</td>
<td>Symmetric</td>
</tr>
<tr>
<td>$S^{-1}$</td>
<td>$\text{Symmetric}(S)$</td>
<td>Symmetric</td>
</tr>
<tr>
<td>expr</td>
<td>expr $==$ expr$^T$</td>
<td>Symmetric</td>
</tr>
</tbody>
</table>

More than a hundred such rules!
### Building blocks:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Constraint</th>
<th>Inferred Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1 + \cdots + S_n$</td>
<td>$\forall_i \text{Symmetric}(S_i)$</td>
<td>Symmetric</td>
</tr>
<tr>
<td>$-S$</td>
<td>$\text{Symmetric}(S)$</td>
<td>Symmetric</td>
</tr>
<tr>
<td>$S^T$</td>
<td>$\text{Symmetric}(S)$</td>
<td>Symmetric</td>
</tr>
<tr>
<td>$S^{-1}$</td>
<td>$\text{Symmetric}(S)$</td>
<td>Symmetric</td>
</tr>
<tr>
<td>$\text{expr}$</td>
<td>$\text{expr} == \text{expr}^T$</td>
<td>Symmetric</td>
</tr>
</tbody>
</table>

More than a hundred such rules!
### Algorithm 4 CLAK-EIG

1. $Z \Lambda Z^T = \Phi$
2. for $i := 1$ to $m$ do
3. \[ K_i := X_i^T Z \]
4. end for
5. for $j := 1$ to $t$ do
6. \[ D_j := h_j \Lambda + (1 - h_j)I \]
7. \[ y_j := Z^T y_j \]
8. for $i := 1$ to $m$ do
9. \[ V_{ij} := K_i D_j^{-1} \]
10. \[ A_{ij} := V_{ij} K_i^T \]
11. \[ Q_{ij} R_{ij} = A_{ij} \]
12. \[ b_{ij} := V_{ij} y_j \]
13. \[ b_{ij} := Q_{ij}^T b_{ij} \]
14. \[ b_{ij} := R_{ij}^{-1} b_{ij} \]
15. end for
16. end for
Derivative Operator

\[ z = \alpha \times x + y \quad \longrightarrow \quad \text{dv}(z = \alpha \times x + y) \ ?:\]

- \[ \text{dv}(z) = \text{dv}(\alpha) \times x + \alpha \times \text{dv}(x) + \text{dv}(y) \]
- \[ \text{dv}(z) = \text{dv}(\alpha) \times x + \alpha \times \text{dv}(x) \]
- \[ \text{dv}(z) = \text{dv}(\alpha) \times x \]
- \[ \ldots \]
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
            [...]
    end
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
1. Two Linear Algebra Compilers

2. CLAK

3. CL1ck

4. Contributions
CL1CK: The Methodology

**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* \((P_{\text{pre}})\) and the *Postcondition* \((P_{\text{post}})\).
Operations are described by means of two predicates: The \textit{Precondition} ($P_{\text{pre}}$) and the \textit{Postcondition} ($P_{\text{post}}$).

\textbf{Example: Derivative of Cholesky}

\[ G = g\text{Chol}(L, B) \equiv \begin{cases} 
    P_{\text{pre}} : \{ \text{Output}(G) \land \text{Input}(L) \land \text{Input}(B) \land \\
    \text{Matrix}(G) \land \text{Matrix}(L) \land \text{Matrix}(B) \land \\
    \text{LowerTriangular}(G) \land \text{Symmetric}(B) \land \\
    \text{LowerTriangular}(L) \} \\
    P_{\text{post}} : \{ GL^T + LG^T = B \} 
\end{cases} \]
CL1CK: The Methodology

CL1CK implements the FLAME methodology in 3 stages:
\[ G = gChol(L, B) \equiv \begin{cases} \begin{align*} P_{\text{pre}} : & \text{Output}(G) \land \text{Input}(L) \land \text{Input}(B) \land \\ & \text{Matrix}(G) \land \text{Matrix}(L) \land \text{Matrix}(B) \land \\ & \text{LowerTriangular}(G) \land \text{Symmetric}(B) \land \\ & \text{LowerTriangular}(L) \end{align*} \end{cases} \]

\[ P_{\text{post}} : \{GL^T + LG^T = B\} \]
\( G = gChol(L, B) \equiv \) 

\[
P_{\text{pre}} : \{ \text{Output}(G) \land \text{Input}(L) \land \text{Input}(B) \land \\
\text{Matrix}(G) \land \text{Matrix}(L) \land \text{Matrix}(B) \land \\
\text{LowerTriangular}(G) \land \text{Symmetric}(B) \land \\
\text{LowerTriangular}(L) \}
\]

\[
P_{\text{post}} : \{ GL^T + LG^T = B \}
\]

\[
\downarrow
\]

equal[


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

\[ GL^T + LG^T = B \]

\[
\begin{pmatrix}
G_{TL} & 0 \\
G_{BL} & G_{BR}
\end{pmatrix}
\begin{pmatrix}
L_{TL}^T & L_{BL}^T \\
0 & L_{BR}^T
\end{pmatrix}
+ \begin{pmatrix}
L_{TL} & 0 \\
L_{BL} & L_{BR}
\end{pmatrix}
\begin{pmatrix}
G_{TL}^T & G_{BL}^T \\
0 & G_{BR}^T
\end{pmatrix}
= \begin{pmatrix}
B_{TL} & B_{BL}^T \\
B_{BL} & B_{BR}
\end{pmatrix}
\]
\[
GL^T + LG^T = B
\]

\[
\downarrow
\]

\[
\begin{pmatrix}
G_{TL} & 0 \\
G_{BL} & G_{BR}
\end{pmatrix}
\begin{pmatrix}
L_{TL}^T & L_{BL}^T \\
0 & L_{BR}^T
\end{pmatrix}
+ \begin{pmatrix}
L_{TL} & 0 \\
L_{BL} & L_{BR}
\end{pmatrix}
\begin{pmatrix}
G_{TL}^T & G_{BL}^T \\
0 & G_{BR}^T
\end{pmatrix}
= \begin{pmatrix}
B_{TL} & B_{BL}^T \\
B_{BL} & B_{BR}
\end{pmatrix}
\]

\[
\downarrow
\]

\[
\begin{pmatrix}
G_{TL}L_{TL}^T + L_{TL}G_{TL}^T = B_{TL} \\
G_{BL}L_{TL}^T + L_{BL}G_{TL}^T = B_{BL} \\
G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T = B_{BR}
\end{pmatrix}
\]
Identifying the subproblems

\[
\begin{align*}
G_{TL}L_{TL}^T + L_{TL}G_{TL}^T &= B_{TL} \\
G_{BL}L_{BL}^T + L_{BL}G_{BL}^T &= B_{BL}
\end{align*}
\]

\[
\begin{align*}
G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T &= B_{BR}
\end{align*}
\]
Identifying the subproblems

\[
\begin{pmatrix}
G_{TL}L_{TL}^T + L_{TL}G_{TL}^T &= B_{TL} \\
G_{BL}L_{TL}^T + L_{BL}G_{TL}^T &= B_{BL} \\
\end{pmatrix}
\]

\[
\Downarrow
\]

\[
\begin{pmatrix}
G_{TL} := gChol(L_{TL}, B_{TL}) \\
G_{BL}L_{TL}^T + L_{BL}G_{TL}^T &= B_{BL} \\
\end{pmatrix}
\]
Identifying the subproblems

\[
\begin{align*}
G_{TL} &:= gChol(L_{TL}, B_{TL}) \\
G_{BL}L_{TL}^T + L_{BL}G_{TL}^T &= B_{BL} \\
G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T &= B_{BR}
\end{align*}
\]
CL1CK: PME Generation

Identifying the subproblems

\( G_{TL} := gChol(L_{TL}, B_{TL}) \)

\[
\begin{align*}
G_{BL}L_{TL}^T + L_{BL}G_{TL}^T &= B_{BL} \\
G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T &= B_{BR}
\end{align*}
\]

\[
\downarrow
\]

\[
\begin{align*}
G_{TL} := gChol(L_{TL}, B_{TL}) \\
G_{BL}L_{TL}^T &= B_{BL} - L_{BL}G_{TL}^T \\
G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T &= B_{BR}
\end{align*}
\]
Identifying the subproblems

\[
\begin{align*}
G_{TL} & := g\text{Chol}(L_{TL}, B_{TL}) \\
G_{BL}L_{TL}^T & = B_{BL} - L_{BL}G_{TL}^T \\
G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T & = B_{BR}
\end{align*}
\]
Identifying the subproblems

\[
G_{TL} := gChol(L_{TL}, B_{TL})
\]

\[
G_{BL}L_{TL}^T = B_{BL} - L_{BL}G_{TL}^T
\]

\[
\begin{align*}
G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T &= B_{BR} \\
\end{align*}
\]

\[
G_{TL} := gChol(L_{TL}, B_{TL})
\]

\[
G_{BL} := (B_{BL} - L_{BL}G_{TL}^T)L_{TL}^{-T}
\]

\[
\begin{align*}
G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T &= B_{BR} \\
\end{align*}
\]
Identifying the subproblems

\[
\left( \begin{array}{c}
G_{TL} := gChol(L_{TL}, B_{TL}) \\
G_{BL} := (B_{BL} - L_{BL}G_{TL}^T)L_{TL}^{-T}
\end{array} \right) \times
\left( 
\begin{array}{c}
G_{BL}L_{BL}^T + G_{BR}L_{BR}^T + L_{BL}G_{BL}^T + L_{BR}G_{BR}^T = B_{BR}
\end{array} \right)
\]
Identifying the subproblems

\[
\begin{align*}
G_{TL} &:= g\text{Chol}(L_{TL}, B_{TL}) \\
G_{BL} &:= (B_{BL} - L_{BL}G_{TL}^{T})L_{TL}^{-T}
\end{align*}
\]

\[
\frac{G_{BL}L_{BL}^{T} + G_{BR}L_{BR}^{T} + L_{BL}G_{BL}^{T} + L_{BR}G_{BR}^{T} = B_{BR}}{G_{BR}L_{BR}^{T} + L_{BR}G_{BR}^{T} = B_{BR} - G_{BL}L_{BL}^{T} - L_{BL}G_{BL}^{T}}
\]
Identifying the subproblems

\[
\begin{align*}
G_{TL} &:= g\text{Chol}(L_{TL}, B_{TL}) \\
G_{BL} &:= (B_{BL} - L_{BL} G_{TL}^T) L_{TL}^{-T} \\
G_{BR} L_{BR}^T + L_{BR} G_{BR}^T &= B_{BR} - G_{BL} L_{BL}^T - L_{BL} G_{BL}^T
\end{align*}
\]
The PME

\[
\begin{align*}
G_{TL} & := gChol(L_{TL}, B_{TL}) \\
G_{BL} & := (B_{BL} - L_{BL}G_{TL}^T)L_{TL}^{-T} \quad \Rightarrow \quad G_{BR}L_{BR}^{T} + L_{BR}G_{BR}^{T} = B_{BR} - G_{BL}L_{BL}^{T} - L_{BL}G_{BL}^{T}
\end{align*}
\]
Triangular Sylvester Equation ($AX + XB = C$):

\[
\begin{pmatrix}
X_{TL} := \Omega(A_{TL}, B_{TL}, C_{TL}) \\
X_{TR} := \Omega(A_{TL}, B_{BR}, C_{TR} - X_{TL}B_{TR}) \\
X_{BL} := \Omega(A_{BR}, B_{TL}, C_{BL} - A_{BL}X_{TL}) \\
X_{BR} := \Omega(A_{BR}, B_{BR}, C_{BR} - X_{BL}B_{TR} - A_{BL}X_{TR})
\end{pmatrix}
\]
Triangular Sylvester Equation \((AX + XB = C)\):

\[
\begin{align*}
X_{TL} &:= \Omega(A_{TL}, B_{TL}, C_{TL}) \\
X_{TR} &:= \Omega(A_{TL}, B_{BR}, C_{TR} - X_{TL}B_{TR}) \\
X_{BL} &:= \Omega(A_{BR}, B_{TL}, C_{BL} - A_{BL}X_{TL}) \\
X_{BR} &:= \Omega(A_{BR}, B_{BR}, C_{BR} - X_{BL}B_{TR} - A_{BL}X_{TR})
\end{align*}
\]

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

\[
\begin{align*}
X_{TL} &:= \Lambda(A_{TL}, C_{TL}) \\
X_{BL} &:= \Omega(A_{BR}, A_{TL}^T, C_{BL} - A_{BL}X_{TL}) \\
X_{BR} &:= \Lambda(A_{BR}, C_{BR} - X_{BL}A_{BL}^T - A_{BL}X_{BL}^T)
\end{align*}
\]
\[
\begin{align*}
X_{TL} & := \Omega(A_{TL}, B_{TL}, C_{TL}) \\
X_{BL} & := \Omega(A_{BR}, B_{TL}, C_{BL} - A_{BL}X_{TL}) \\
X_{TR} & := \Omega(A_{TL}, B_{BR}, C_{TR} - X_{TL}B_{TR}) \\
X_{BR} & := \Omega(A_{BR}, B_{BR}, C_{BR} - X_{BL}B_{TR} - A_{BL}X_{TR})
\end{align*}
\]
**Partition**  
\[ B \rightarrow \left( \begin{array}{c|c|c} B_{TL} & \ast & \ast \\ \hline B_{BL} & B_{BR} & \end{array} \right) , \quad L \rightarrow \left( \begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array} \right) , \quad G \rightarrow \left( \begin{array}{c|c|c} G_{TL} & 0 \\ \hline G_{BL} & G_{BR} \end{array} \right) \]

where \( B_{TL}, L_{TL}, \) and \( G_{TL} \) are \( 0 \times 0 \)

**while**  
\[ \text{size}(B_{TL}) < \text{size}(B) \text{ do} \]

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

**Variant 1**

\[
G_{10} := B_{10} - L_{10} G_{00}^T
\]

\[
G_{11} := B_{11} - G_{10} L_{10}^T - L_{10} G_{10}^T
\]

\[
G_{11} := \text{gChol}(G_{11}, L_{11})
\]

**Variant 2**

\[
G_{10} := G_{10} L_{00}^{-T}
\]

\[
G_{11} := B_{11} - G_{10} L_{10}^T - L_{10} G_{10}^T
\]

\[
G_{11} := \text{gChol}(G_{11}, L_{11})
\]

**Variant 3**

\[
\ldots
\]

**Variant 4**

\[
\ldots
\]

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

**endwhile**
void gChol_blk_var1( FLA_Obj G, FLA_Obj L, int nb )
{
    FLA_Obj GTL, GTR, GBL, GBR, GO0, 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, GO0, 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)
  - ...
Two Linear Algebra Compilers

CLAK

CL1CK

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
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
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 **

** CL1ck **
Computational Biology


Fabregat & Bientinesi. Computing petaflops over terabytes of data: 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
Future research directions

- 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.